UNIVERSITY OF PRIMORSKA
FACULTY OF MATHEMATICS, NATURAL SCIENCES AND INFORMATION TECHNOLOGIES

Milan Đorđević

GRAFTED GENETIC ALGORITHM AND THE TRAVELING VISITOR PROBLEM

PhD Thesis

August 2012

Supervisor: Prof. Andrej Brodnik, PhD
Co-Supervisor: Prof. Janez Žibert, PhD

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my Director of Studies, Prof. Andrej Brodnik. He has helped, guided and supported me throughout this work. I am deeply indebted to Andrej for his generous and invaluable advice on various matters, be it writing papers, giving talks, preparing slides, or other non-research matters. I am particularly thankful for his patience and encouragement, which has allowed me to pursue what has interested me.

I especially want to thank my Co-Supervisor: Prof. Janez Žibert, whose support and guidance was crucial for successful completion of my thesis work. Janez has been vitally involved in essential last months of my work and has constantly been available to instruct me.

I also thank the other two members of my thesis committee: Dr. Branko Kavšek and Dr. Zoran Bosnić, whose practical advice enlarged legibility and reduced ambiguity of the thesis.

I would like to acknowledge the financial support of the Slovene Human Resources Development and Scholarship Fund "AD FUTURA", especially in Postgraduate Studentship that provided the funding for this research.

I am also very grateful and extend my sincere thanks to Dr. Dragan Marušič, Dr. Klavdija Kutnar, Dr. Štefko Miklavič, Dr. Iztok Savnik, Dr. Martin Milanič and Dr. Bojan Kuzma for their rewarding discussion and unselfish support.

My gratitude is also deserved by the members of the DIST research group from UP FAMNIT, who have provided help and friendship during my time in the group. My particular thanks go to Jernej, Tine, Andrej, Damjan, David, Matjaž and Marko. My thanks go to the helpful and friendly members of department of mathematics from UP FAMNIT and in particular to Cui, Enes, Boštjan, Ademir, Samed and Pavel.

A special thanks goes to the other colleagues and scientist in OR, EC, Economics, Organizations, Maths and Psychology for sharing with me their ideas, insights, views, projects, etc. during PhD studies. I would like to give thanks to Dr. William Cook, Dr. Peter Bosman, Dr. Marko Urh, Mr. Rado Pezdir, Dr. Gordana Komazec, Dr. Dragan Urošević and Dr. Milan Tuba.

I would also like to thank all my best friends, particularly Tadija, Pop, Neša, Crni, Irena, Irman, Andreja, Tomaž, Peđa, Boštjan and Mlađa. You helped make my university life most enjoyable.

Thanks to my family in Belgrade: Dada, Daja, Olja, Dace, Peđa, Goran, Kaća, Jeca, Gaga, Boki, Sneža, Goga, Tića, Saška, and their families. I am very grateful for your support.

To the woman that is my life companion, my girlfriend Jasmina Simović. You bring laughter, tenderness and, above all, love to my life. Therefore, "thank you" are too small words to express gratitude.

Finally and most importantly, I thank my family for their love, and constant support and encouragement. I want to thank my father Gmitar Đorđević, my mother Slavica Đorđević and my brother Bojan Đorđević for the moral and spiritual harmony they represent for me.

Dedicated to my family!

Abstract

Grafted Genetic Algorithm and the Traveling Visitor Problem

In this PhD Thesis, two related topics from theoretical computer science are considered. The first one is hybridization of genetic algorithms (Grafted Genetic Algorithm). The thesis analyzes separate, combined and partial performance of genetic algorithm hybridization with local searcher. On the Traveling Salesman Problem we examine the impact of hybridization a 2-opt heuristic-based local searcher into the genetic algorithm. Genetic algorithm provides a diversification, while 2-opt improves intensification. Results on examples from TSPLIB show that this method combines good qualities from the both applied methods and outperforms each individual method. In tests we applied hybridization at various percentages of genetic algorithm iterations. On one side, the less frequent application of hybridization decreased the average running time of the algorithm from 14.62 sec to 2.78 sec at 100% and 10% hybridization respectively, while on the other side the quality of solution on average deteriorated only from 0.21% till 1.40% worse than the optimal solution. We also studied at which iterations of the genetic algorithm to apply the hybridization. We applied it at random iterations, at the initial iterations, and the ending ones where the later proved to be the best.

The second topic considered is the Traveling Visitor Problem. In the problem, visitor starts from a hotel to visit all interesting sites in a city exactly once and to come back to the hotel. Since, the visitor uses streets and pedestrian zones, she can not cut the corners. The goal is to minimize the visitor's traveling distance. This problem is similar to the Traveling Salesman Problem. The difference is that during the visit, traveling visitors must go around buildings. This means that shortest distances, like those in Euclidean Traveling Salesman Problem, are not valid. The tested benchmarks come from three real instances of cities Venice, Belgrade and Koper and two artificial cases made of TSPLIB. We introduced and compared two methods for solving the Traveling Visitor Problem. In all tested cases, the Koper Algorithm outperforms the Naïve Algorithm for solving the Traveling Visitor Problem - quality of solutions differs from 6.52% to 354.46%.

ACM Computing Classification System: F.2.1, G.1.2, G.1.6, G.2.2, G.2.3.

Key words: Traveling Salesman Problem, Genetic Algorithms, Hybrid Genetic Algorithms, Memetic Algorithms, Grafted Genetic Algorithms, Traveling Visitor Problem.

Izvleček

Cepljeni genetski algoritmi in problem potujočega obiskovalca

V doktorski disertaciji sta obravnavani dve med seboj povezani temi s področja teoretičnega računalništva:

Prva tema obravnava cepljene (hibridizirane) genetske algoritme. V disertaciji je analizirana izvedba lokalnega iskanja ter genetskega algoritma. Na problemu trgovskega potnika (angl. Traveling Salesman Problem) smo preučevali vpliv hibridizacije 2-opt hevrističnega lokalnega iskalca v genetski algoritem. Zadnji zagotavlja raznolikost (angl. diversification) , medtem ko 2-opt izboljša krepitev (angl. intensification). Rezultati primerov iz TSPLIB kažejo na to, da metoda združuje dobre lastnosti obeh metod in prekaša vsako posamezno metodo. V testih smo uporabili hibridizacijo pri različnih odstotkih genetskih iteracij algoritma. Z manjšanjem odstotka hibridizacije se zmanjšuje povprečni čas izvedbe algoritma, ki znaša 14,62 s pri 100 % hibridizaciji, in le 2,78 s pri 10 % hibridizaciji. Po drugi strani pa se kakovost rešitve pri enakih odstotkih hibridizacije poslabša samo z 0,21 % na 1,40 % odstopanja od optimalne rešitve. Prav tako smo preizkušali, katere iteracije genetskega algoritma je treba hibridizirati. Hibridizacijo smo izvedli pri naključnih, začetnih in končnih iteracijah, za katere se je kasneje izkazalo, da so najboljše.

Druga tema disertacije obravnava problem potujočega obiskovalca (angl. Traveling Visitor Problem). V tem problemu si obiskovalec želi ogledati vse zanimive lokacije v mestu natanko enkrat in se po ogledu vrniti v hotel. Cilj je skrajšati dolžino sprehoda potujočega obiskovalca in se pri tem izogniti oviram (stavbam) na poti, slednje pomeni, da pristop, kjer bi problem reševali kot evklidski ravninski problem trgovskega potnika ni sprejemljiv, saj se slednji ne izogiba oviram. Za testiranje smo uporabili primere problema, ki smo jih izdelali na osnovi zemljevidov mest: Kopra, Beograda in Benetk. Poleg tega smo iz knjižnice TSPLIB izbrali dva problema ter ju preoblikovali v problema potujočega obiskovalca. Pri reševanju problema potujočega obiskovalca smo primerjali dve metodi. V vseh testiranih primerih problema algoritem koper prekaša naivni algoritem za reševanje problema potujočega obiskovalca - razlika v kakovosti rešitve znaša od 6,52 % do 354,46 %.

ACM Computing Classification System: F.2.1, G.1.2, G.1.6, G.2.2, G.2.3.

Ključne besede: problem trgovskega obiskovalca, genetski algoritmi, hibridni genetski algoritmi, memetični algoritmi, cepljeni genetski algoritmi, problem potujočega obiskovalca.

Contents

List of Figures

List of Algorithms

List of Tables

Chapter 1

Introduction

I arrived to Koper, from Belgrade, on postgraduate studies of computer science, in 2007. Immediately, I had an urge to acquaint myself with the city in which I was to live for the next four years. I was drawn by the city center with its numerous sites - exactly 55 of them were listed in the official tourist map of Koper. Since the doctoral studies were exhausting, I did not have much spare time to roam the streets of Koper. So I began to wonder whether I could devise a way of optimizing the tour through the city, by visiting all sites in as few steps as possible. We named the problem the *Traveling Visitor Problem* (TVP).

1.1 Traveling Visitor Problem

The Traveling Visitor Problem is a version of the *Traveling Salesman Problem*, (TSP) [71, 94], with a difference that the traveling visitors, during their visit of sites, cannot fly over the buildings in the city, instead visitors must go around these obstacles. This difference is demonstrated in Figure 1.1. This means that direct edge connections, as we know them in the TSP, are in this case impossible (direct edge

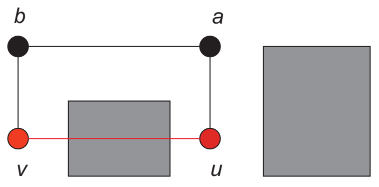

Figure 1.1: A simple case of TVP: Two rectangles represent buildings (obstacles) in the city. Red nodes represent sites of interest in the city (vertices from set S), black nodes represent crossroads in the city (vertices from set X), the red line represent the shortest connection between two interesting sites, black lines represent the connections between two sites of interest, going via two crossroads.

from u to v in Figure 1.1). Visitors use the walking paths and pedestrian zones of variable length. These limits determine the weight of edges connecting the vertices in the graph.

Formally, the Traveling Visitor Problem is stated as: given a connected, weighted graph $G = (V, E, W)$, with a set of vertices $V = S \cup X$ and $S \cap X = \emptyset$, S belongs to interesting sites in a city, X belongs to crossroads in a city, a set of edges E, and a cost of traveling W. The goal is to find the shortest closed walk through all vertices of S, according to W in graph G, although we may walk through vertices from X.

Chapter 2 introduces the basic concepts of the Traveling Salesman Problem and related problems. Furthermore, the algorithms for solving it are described. This chapter provides the knowledge necessary for further chapters of the dissertation to become understandable to the broad range of readers. Furthermore, it presents an interpretation of the various textbooks which represents the contemporary literature on related areas of computer science.

1.2 Traveling Salesman Problem

In the early 30's of the 20th century, the Austrian mathematician Karl Menger challenged the research community of that time to consider, from the mathematical point of view, the following problem: A traveling salesman has to visit exactly once each one of a list of n cities and then return to the home city. He knows the cost of traveling from any city i to any other city j. Thus, the question is, which is the tour of least possible cost the salesman can take [94].

In the TSP a set $\{c_1, c_2, ...c_n\}$ of cities is considered and for each pair (c_i, c_j) where $i \neq j$, a distance $d(c_i, c_j)$ is given. The goal is to find a permutation π of the cities that minimizes the quantity

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}). \tag{1.1}$$

This quantity is referred to as the tour length since it is the length of the tour a salesman would have to travel when visiting the cities in the order specified by the permutation π, returning at the end to the initial city.

TSP is one of the most important representatives of a large class of problems known as combinatorial optimization problems [63]. Since the TSP belongs to a class of *NP-hard* problems [72], an efficient algorithm for TSP probably does not exist. More accurately, such an algorithm exists if and only if the two computational classes P and NP coincide. From a practical point of view, it means that in general it is quite impossible to find an exact solution for any TSP instance with n nodes that has a behavior considerably better than the algorithm which computes all of the $(n-1)!$ possible distinct tours, and then returns the least costly one.

If we are looking for applications, a different approach can be used. Given a TSP instance with n nodes, any tour passing once through all cities is a feasible solution. Algorithms that construct in polynomial time with respect to n feasible solutions are called heuristics [12,115]. In general, these algorithms produce solutions but without any insights to how far is their cost from the optimal.

There exist two versions of TSP: the Symmetric and the Asymmetric TSP. In the symmetric form, known as the STSP [88], the cost distance between nodes i and j is equal to the cost distance between nodes j and i ($c_{ij} = c_{ji}$). Comprehensive surveys on STSP can be found in [69,70,92] In the case of asymmetric TSP (ATSP) [18], there is no such symmetry. Comprehensive surveys on ATSP can be found in [14, 16, 23] In addition, there are many different variations of TSP which are described and explored in the literature and also a variations derived from everyday life. From the dissertation point of view, the most important applications of the TSP studied in the literature, are in more details presented in Chapter 2.

First steps in solving the TSP consist of exact methods and heuristics. Exact methods like cutting planes [109] and branch and bound [26,109], can optimally solve relatively small problems (depending on the size of n), while methods such as different variants of Lin-Kernighan algorithm [6, 45, 66, 75], and Concorde algorithm [3–5] are good for larger problems. Furthermore, some algorithms based on greedy principles such as nearest neighbor [60], and spanning tree [65] can be also used for solving the TSP. The above-mentioned methods for solving TSP result in exponential computational complexities. For that reason a new methods are required to overcome this shortcoming. These new methods include different kinds of heuristic techniques, nature based optimization algorithms, etc. Various creatures and natural systems, developed in nature, are interesting and valuable sources of inspiration for exploring and creating new methods for solving the TSP and variations of TSP. Let us count the most important of them. Evolutionary Algorithms [7], Genetic Algorithms [67], Memetic Algorithms [84], Simulated Annealing [80], Ant Systems [38] and finally Grafted Genetic Algorithms [34–37]. The latter is a type of hybrid genetic algorithms [42, 64, 89, 133] and they will be described and demonstrated in the thesis. The first topic of the thesis, presented in Chapter 3, deals with the heuristic method called the Grafted Genetic Algorithm, through which we solve the Traveling Salesman Problem. In this chapter the aim is to show the quality of the solution and the running time of the grafted genetic algorithm when it is applied to the instances of the problems of symmetric TSP which are available on the Internet.

The second topic of the thesis, presented in Chapter 4, elaborates the Traveling Visitor Problem. This chapter describes a problem and examples of real cities and finally solves the instances of the problem by using new methods.

1.3 Research Objectives

Research objective of the thesis is to prove the following hypotheses:

- Hypothesis 1: The method for solving of TSP that is made of two independent methods, genetic algorithm and 2-opt heuristic, outperforms each of the combined methods in terms of the quality of solution.
- Hypothesis 2: The quality of solution of a proposed method to the problem

of a traveling visitor problem, outperforms algorithms for solving general TSP problem, when they are used for solving traveling visitor problem.

Proofs of hypothesis 1 and the hypothesis 2 are in the Chapter 3 and Chapter 4, respectively.

Contributions to the science consist of the following results:

- construction of the grafted genetic algorithm for solving the traveling salesman problem,
- insights of different levels and places of hybridization for grafted genetic algorithm,
- construction of the method for solving the traveling visitor problem,
- construction of improved Floyd-Warshall algorithm for all-pairs shortest path problem,
- solutions of the traveling visitor problem for cities of Koper, Belgrade and Venice.

The results of this PhD Thesis are published in the following articles:

- M. Djordjevic, Influence of Grafting a Hybrid Searcher Into the Evolutionary Algorithm, *Proceedings of the 17th International Electrotechnical and Computer Science Conference, Portoroz, Slovenia* (2008), 115–118.
- M. Djordjevic, M. Tuba, and B. Djordjevic, Impact of Grafting a 2-opt Algorithm Based Local Searcher Into the Genetic Algorithm, *Proceedings of the 9th WSEAS international conference on Applied informatics and communications, AIC 2009, Moscow, Russia* (2009), 485–490.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Book of Abstract of International Conference on Operations Research, OR 2011, Zurich, Switzerland* (2011), 130.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011, Dubrovnik, Croatia* (2011), 515–520.
- M. Djordjevic, M. Grgurovic and A. Brodnik, The Traveling Visitor Problem and the Koper Algorithm for Solving It, *Book of Abstracts of 25th Conference of European Chapter on Combinatorial Optimization, ECCO 2012, Antalya, Turkey* (2012), 10.
- M. Djordjevic, M. Grgurovic and A. Brodnik, The Traveling Visitor Problem and Algorithms for Solving It, *Book of Abstracts of 3rd Student Conference on Operational Research, SCOR 2012, Nottingham, UK* (2012), 26.
- M. Djordjevic, J. Zibert, M. Grgurovic, and A. Brodnik Methods for Solving the Traveling Visitor Problem, *Proceedings of the 1st International Internet and Business Conference, IBC 2012, Rovinj, Croatia* (2012), 174–179.

- M. Djordjevic, M. Grgurovic, and A. Brodnik, Performance Analysis of Partial Use of Local Optimization Operator on Genetic Algorithm for Traveling Salesman Problem, *Business Systems Research*, Print ISSN 1847-8344; Online ISSN 1847-9375, in press.

Chapter 2

Background

2.1 Traveling Salesman Problem

In this chapter we introduce the basic concepts of optimization and related combinatorial problems. Among those problems, probably best known is the Traveling Salesman Problem (TSP) , which we describe in this section in more detail. We illustrate the computational properties of the TSP which is closely related to many optimization problems arising in real world applications. We refine some graph theoretical approaches, which can be applied when attempting to tackle traveling salesman like problems. TSP can be stated as follows. Including his home town, a salesman wants to visit n cities and then return home. The objective is to find a tour for visiting each city exactly once while minimizing the total distance traveled. The problem of finding a minimal tour is equivalent to finding an optimal ordering of the set of cities. The TSP is consequently often formulated as a permutation problem. Given a set of n cities $\{c_1, c_2, ..., c_n\}$ and for each pair of cities (c_i, c_j) a distance $d(c_i, c_j)$. The goal is then to find a permutation π of the cities which minimizes the length of the tour given by:

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}). \tag{2.1}$$

As mentioned before, the TSP has attracted the focus of researchers for decades. According to a survey on the history of combinatorial optimization by [118], the TSP was formulated as early as 1932 in a German manual for the successful traveling salesman, before it was presented as a research problem by Menger in the 1930's [94]. Since then, the TSPs simplicity on the one hand and the difficulty of finding optimal solutions on the other, has affirmed it as a test bed for new heuristics and exact algorithms. But, the problem has not only theoretical foundations and there are many real life applications of the TSP and its variants [74, 108], some of which we will describe later in this chapter. Let us now introduce the TSP in mathematical terms as an integer programming problem. Take up variables $x_{i,j}$ which are either equivalent to one if the city j is straightly visited after city i or zero otherwise. The TSP may be established as an integer program [79] as follows:

$$\begin{aligned}
\text{Minimize } & \sum_{i=1}^{n}\sum_{j=1}^{n} d(c_i,c_j)x_{i,j}, \\
\text{Subject to } & \sum_{i=1}^{n} x_{i,j} = 1, \forall j = 1,2,...,n, \\
& \sum_{j=1}^{n} x_{i,j} = 1, \forall i = 1,2,...,n, \\
& \sum_{i\in I}\sum_{j\notin I} x_{i,j} \geq 1, I \subset \{1,2,...,n\} \text{ with } 1 < |I| < n, \\
\text{and } & x_{i,j} \in \{0,1\}, \forall i,j = 1,2,...,n,
\end{aligned} \tag{2.2}$$

Here, our goal is to give the sum of all combination variables $x_{i,j}$ weighted by the lengths between the cities. This function is subject to a number of constraints. The constraints stated first and second inflict each city to be visited exactly once as a part of the tour. The constraints denoted third are the so called sub-tour avoidance constraints which are needed to guarantee that the solution represents a connected tour. The presentation of the TSP as an integer program shown above is not unique; there are various representations, see for example [106]. In the next subsections we will present some important notions for optimization problems and the TSP, together with a review of problems similar to the TSP.

2.1.1 Graph representation of the TSP

TSP is ordinarily represented and considered as a graph theoretical problem [74, 79]. This representation is suitable because many algorithms for the TSP are established on graph theoretical concepts. Here, we present some graph theoretical definitions needed to represent the TSP in graph theoretical terms.

An undirected graph $G = (V, E)$ consists of a finite set of vertices (or nodes) V and a finite set of edges E. Each edge $e \in E$ correspond to a set $e = \{u, v\}$ of two vertices $u, v \in V$. An edge $e = \{u, v\}$ is *incident* to vertices u and v and the number of edges incident to a vertex is the *degree* of the vertex. In the situation where there exists an edge $e = \{u, v\}$ between each pair $u, v \in V$ we speak of a *complete graph*. Now, some concepts of the TSP will be presented. If $P = (\{u_1, u_2\}, \{u_2, u_3\}, ..., \{u_{k-1}, u_k\})$, where $\{u_i, u_{i+1}\} \in E$, then P is called a *walk*. If further $u_i \neq u_j$ for all $i \neq j$ it is called a *path*. A closed path, where $P = (\{u_1, u_2\}, \{u_2, u_3\}, ..., \{u_k, u_1\})$, i.e. a path as well as the edge returning to the vertex it started from, is denoted to as a *cycle*. Graphs are generally categorized according to their properties. Two important classes, for the TSP are the *Eulerian* and the *Hamiltonian graphs*. A graph is called *Eulerian* if it contains an *Eulerian Tour*, which is a closed walk traversing every edge of the E in graph G. A graph is called *Hamiltonian* if there exists a cycle which visits every vertex from V in graph

G. Those cycles are called *Hamiltonian Cycles.* It is worthy to measure the qualities of a walk, path or tour in comparison to others. Furthermore, *weights* are associated with each edge of the graph. We denote such a graph a *weighted graph.* Because a walk and path can be identified by the set of edges traversed, P, its quality can be valuated by a weight or cost function $c : P \longrightarrow \mathbb{Q}$, which will map the edge set P into the set of the rational numbers $\mathbb{Q}$. Let $w_{i,j}$ refer to the weight of edge $\{i, j\}$, then the weight of an edge set P is then specified as the sum over all the edges of P:

$$\sum_{\{i,j\}\in P} w_{i,j}. \tag{2.3}$$

For a tour or a path the sum of weights of its edges is commonly specified to as its *length.* Let us now determine the symmetric TSP (STSP) [69, 70, 88, 92]. In the STSP the distance between nodes i and j is equal to the distance between nodes j and i. An instance of a TSP can be seen as a complete graph $G = (V, E)$ where the set of vertices V is given by the cities and edges in the graph with corresponding edge weights $c_{i,j}$ are given by the distances between cities. The TSP is then equal to the problem of finding a Hamiltonian Tour of minimal length in the graph G. For many applications it is useful to correlate a direction of the edges of a graph. Because the weights between vertices are not necessary symmetric, i.e. $c_{i,j} \neq c_{j,i}$, for some edges of G. This kind of graph is called *directed graph* or *digraph* and its edges are ordered 2-tuples of vertices, referred to as *arcs.* The asymmetric TSP (ATSP) [14, 16, 18, 23] is then similar to the symmetric TSP above, i.e. it is the problem of finding a Hamiltonian Tour of minimal length in a complete digraph which also take the directions of the edges into account. The *Euclidean TSP*, or planar TSP, is the TSP with the distance being the ordinary *Euclidean distance.* The Euclidean TSP [55, 107] is then a particular case of metric TSPs, since distances in a plane obey the triangle inequality.

2.1.2 Computational complexity of the TSP

As was mentioned at the start of this section the TSP is a very hard combinatorial optimization problem. In order to evaluate algorithms according to their computational requirements the theory of *Computational Complexity* has been developed. In plain terms, the computational complexity deals with the number of computational steps an algorithm needs to solve an instance of a problem of size n. An algorithm can be described as a "step-by-step procedure" [72]. The number of computational steps realized present a measure of the execution time needed to solve an instance of a problem. The number of steps an algorithm requires, is often not only dependent on the size of the problem instance. It may also differ between instances of the same size. Furthermore, it is not always straightforward to estimate the number of steps an algorithm needs for a given instance. To be able to analyze the complexity of an algorithm, a *worst-case analysis* is introduced. The computational complexity is defined as the *maximum number of steps* an algorithm may require for any instance of a given size.

To recap, the performance of an algorithm is measured as the maximum number of steps it requires for any problem instance of size n denoted as a function of

n. Widely adopted concept is that an algorithm is effective if its worst-case complexity is bounded by a polynomial function of instance size n while algorithms of exponential time complexity are commonly considered ineffective or computationally expensive [72]. Even though an exponential function may initially give smaller values than a given polynomial, there will be always a constant N such that for all $n \geq N$ the polynomial function get lesser values than the exponential.

The Classes P and NP

The outline summarized in previous section can be used to categorize an algorithm as efficient or not dependent upon whether it has polynomial time complexity. Similarly a problem is considered easy or hard depending whether there exists a polynomial time algorithm for solving it. Furthermore, it means that a problem is considered easy if there is an algorithm where in the worst case the number of steps of the problem of size n is bounded by a polynomial of n. Decision Problems consist of an instance of a problem and a question to which the answer is 'yes' or 'no'. An example of a decision problem related to the TSP is the *Hamiltonian Cycle Problem* [72]:

Problem 2.1.1 *Instance: Graph $G = (V, E)$.*

Question: Does there exist a cycle in G passing through each vertex in V exactly once?

As we described above, looking from the graph theoretical point of view, the TSP is equivalent to finding a Hamiltonian cycle of a minimum length in a complete graph G. As a consequence, for the TSP decision problem the main question is not whether a cycle exists in a complete graph G, but if there exists a cycle of length less than a given constant B. The TSP Decision Problem can be stated as follows [72]:

Problem 2.1.2 *Instance: Given a complete weighted graph $G = (V, E)$ with non-negative edge weights ω_i, for $i \in E$ and a constant $B \geq 0$.*

Question: Does there exist a Hamiltonian cycle in G (or a Tour of a Traveling Salesman) with edge sequence S where $\sum_{i \in S} w_i \leq B$?

It is easy to see that if there exists a polynomial time algorithm for the TSP, there also exists a polynomial time algorithm for the TSP Decision Problem. On the other hand, it is not so evident and even not always true that the converse holds as well. A polynomial time algorithm for generating the optimal TSP tours by calling a subroutine that solves the TSP Decision Problem is introduced by Johnson and Papadimitriou in [72]. Supposing that the TSP decision problem can be solved in polynomial time, this algorithm, provides a polynomial time algorithm for the Traveling Salesman Problem. This represents a type of polynomial time reduction [32]. Additionally the presence of this algorithm proves the following theorem [72]:

Theorem 2.1.3 *There exists a polynomial time algorithm for the TSP if and only if there exists a polynomial time algorithm for the TSP Decision Problem.*

This theorem indicates that when we are interested in the computational complexity it is adequate to restrain focus to the TSP Decision Problem. That is why the question of whether the TSP is a "hard" to solve problem is equivalent to whether the TSP Decision Problem is a "hard" problem. Let us now introduce the two classes of problems in the theory of computational complexity. The class P contain all decision problems for which exists a polynomial time algorithm. Many combinatorial optimization problems are not in P. Nevertheless, it is not easily shown that certain problems are not in P even if no polynomial time algorithm is known. To specify the second class of algorithms we must first define the concept of *non-deterministic* algorithms.

Opposite to the deterministic algorithm, a non-deterministic algorithm can exhibit different behaviors on different runs. The class NP are composed of all decision problems, which can be solved in polynomial time by an non-deterministic algorithm. It is obvious that $P \subseteq NP$. In spite of that, the question of whether $P = NP$ denote one of the major question of the computer science community. A lot of research has been spent on this challenge over the last five to six decades and the fact that no polynomial-time algorithm for specific problems in NP has been found, makes the equivalence of both classes very unlikely. Because of this, it is widely assumed that $P \neq NP$, even though this conjecture remains unproven to present date.

The Class NP-Complete

Very important subset of problems in NP is the class of $NP-complete$ problems. NP-complete (Non-deterministic Polynomial time complete). This term represents a property of computational decision problems which is a subset of NP (i.e. can be solved by a non-deterministic Turing Machine in polynomial time). The most important property of NP-complete decision problems is that their computational status is directly connected to the relation between P and NP. The concept of NP-completeness was introduced in [25]. Regrettably, concerning the TSP, it has been shown that it belongs to the class of NP-complete problems. As Johnson and Papadimitriou comment in [72]:

> *This is our main negative result for the TSP; it is the strongest negative result one can hope to prove, short of establishing $P \neq NP$.*

In contrast to the consideration that the TSP is not NP- complete as it is not a decision problem. Earlier in the chapter we have mentioned a polynomial time algorithm for the TSP which directly leads to a polynomial time algorithm for the TSP Decision Problem and also its NP-completeness imply that $P = NP$. Problems which have this property are referred to as *NP-hard* problems.

2.1.3 Applications of the TSP

The TSP variations and the applications of the TSP exceed the route planning problem of a traveling salesman and cross over some areas of science including computer science, mathematics, operations research, genetics, engineering, and electronics. In the next three subsections we summarize, from the dissertation point of view, the most important applications of TSP studied in the literature.

Machine Scheduling Problem

Maybe the most studied application field of the TSP is scheduling and machine sequencing [74]. A scheduling can be described as follows. There are n assignments $\{1, 2, ..., n\}$ to be processed consecutively on a machine. Let $c_{i,j}$, be the setup cost necessary for processing assignment j instantly after assignment i. When all assignments are processed, the machine is reset to its original state at a cost of $c_{j,l}$, where j is the last assignment processed [74]. Therefore the machine sequencing problem is about finding an order in which the assignments are to be processed so that the total setup cost is minimized. Obviously, finding a permutation π of $\{1, 2, ..., n\}$ that minimizes Equation 2.4 solves the problem:

$$c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)} \tag{2.4}$$

Looking from the practical point of view the assignments can be often clustered together. In this case the setup time, if any, between assignment within a cluster is relatively small compared to setup time between jobs in two different clusters. For example this can be a typical scenario in an assembly line. Assembling the similar products needs minimal setup time. However, if a different product needs to be assembled, the setup time may grow larger because it may require the new parts, tools etc. So the machine sequencing problem with such a specialized matrix of costs reduces to the *Clustered TSP* [61]. This problem will be, in more detail, presented in dissertation.

Let us take a look at another scheduling problem - a *no wait flow shop* problem. There are n assignments each demanding processing on m machines in the order $1, 2, 3, ..., m$. No assignment is approved to take a waiting time between processing on two sequential machines. The goal is to find an optimum sequencing of assignments which will be processed so that the total completion time is minimal. Applications of this kind of sequencing problems can arise in a different situations, more details can be found at [43]. Furthermore, the no-wait flow shop problem is strongly NP-hard, but solvable in polynomial time, when $m = 2$. In this particular case it reduces to the well known *Gilmore-Gomory* problem [60].

For details of other research works on machine scheduling problems that are relevant to the TSP, but go beyond the scope of the dissertation, refer to [8, 60].

General Routing Problem

Given a connected, undirected graph $G = (V, E)$, that consists of a finite set of vertices V and a finite set of edges E, a cost c_e for each edge $e \in E$, a finite set $V_R \subseteq V$ of required vertices and a finite set $E_R \subseteq E$ of required edges. The General Routing Problem (GRP) is the problem of finding a minimum cost route passing through each $v \in V_R$ and each $e \in E_R$ at least once [105].

The GRP contains several other important routing problems as special cases:

- The *Rural Postman Problem* (RPP) is obtained when $V_R = \emptyset$. For more details on this problem, we refer to [105].

- The *Chinese Postman Problem* (CPP) is obtained when $V_R = \emptyset$ and $E_R = E$. For more details on this particular problem, we refer to [24].
- The *Steiner Graphical TSP* (SGTSP) is obtained when $E_R = \emptyset$. The SGTSP has been discussed in [28]. Moreover, in [46] this problem was also called the *Road TSP* (RTSP).
- The *Graphical TSP* (GTSP) is obtained when $E_R = \emptyset$ and $V_R = V$. For more details we refer to [28].

Frequency Assignment Problem

If we look a communication network together with a set of transmitters, *the Frequency Assignment Problem* [47] is to specify a frequency to each transmitter from a given set of usable frequencies. The frequencies must satisfying some interference constraints. These constraints can be denoted by a graph $G = (V, E)$ in which each node denotes a transmitter. A non-negative integer weight $c_{i,j}$ is appointed for each arc (i, j) denoting the tolerance. Take $F = \{0, 1, 2, ..., R\}$ to be a collection of usable frequencies. A frequency assignment is then to assign the number $f(i) \in F$ to node $i \in V$ such that $|f(i) - f(j)| > c_{i,j}$ for all $(i, j) \in E$. In the case that such an assignment exists then it is called a feasible assignment. Note that if R is large enough a feasible assignment is possible. Let us consider the following, let G^* be the complete graph obtained from G by adding edges of zero weight. Let $c'_{i,j}$ be the weight of edge (i, j) in G^* such that $c'_{i,j} = c_{i,j} + 1$. Finally, let $C'(H^*)$ be the sum of the weights of edges in a minimal cost Hamiltonian path in G^*. Then the TSP can be used to compute a lower bound for the frequency assignment problem, as it has been shown in [60].

2.1.4 Most Important Variations of the TSP Studied in the Literature

Our discussion on a couple of further variations are confined to their definitions only and practical implementations for some variations. For more comprehensive overview of these particular problems, see the corresponding references cited. Below we summarize, from the dissertation point of view, the most important variations of TSP studied in the literature.

The time dependent TSP: For each arc (i, j) of graph G, n of different costs $c^t_{i,j}, t = 1, 2, ..., n$ are given. The cost $c^t_{i,j}$ represent the cost of traveling from city i to city j in some time period t. The goal is to find a tour $(\pi(1), \pi(2), ...\pi(n), (1))$, where $\pi(1) = 1$ represents the home location, which is in the time period zero, in graph G such that $\sum_{i=1}^{n} c^i_{\pi(i),\pi(i+1)}$ is minimized. The index $n + 1$ is equivalent to 1. When all arcs (i, j), are $c^1_{i,j} = c^2_{i,j} = \ldots = c^n_{i,j}$ then this problem reduces to the TSP. For more details on this particular problem, we refer to [57].

Black and White TSP: The black and white TSP is a generalization of the TSP. In this problem, the set of nodes of G is partitioned into two sets, B and W. The elements of set B are addressed as black nodes and the elements of set

W are addressed as white nodes. For tour H in G to be feasible the following two conditions must be satisfied. Firstly, the number of white nodes between any two sequential black nodes should not go beyond a positive integer I. Secondly, the distance between any two sequential black nodes should not go beyond a positive real number R. Then, the goal of the black and white TSP is to find a minimal cost feasible tour in graph G. It is important to note the practical implementation of this problem. Accordingly, the applications of this problem involve design of ring networks in telecommunications [60]. Furthermore, a variation of this problem known as TSP with replenishment arcs, has been applied in the air-line industry. The TSP with replenishment arcs has been discussed in [93].

The delivery man problem: The delivery man problem (DMP) is also known as the *minimum latency problem* [11] and *the traveling repairman problem* [52]. Let H be a tour in G and v_1 be a starting node. For each vertex v_i of G, define the *latency* of v_i with respect to H expressed by $L_i(H)$, which represents the total distance in H from v_1 to v_i. The goal of the delivery man problem is then to find a tour H^* in graph G such that $\sum_{i=1}^{n} L_i(H^*)$ is minimized. Additionally it can be verified that this problem is a special case of the time dependent TSP described above. For more details on this problem, we refer to [60].

Clustered TSP: In this variation of TSP, the set of vertices of graph G is splitted into clusters $V_1, V_2, ..., V_k$. The goal of clustered TSP tries to find a minimum cost tour H in graph G with the constraint that vertices within the same cluster must be visited consecutively. Note also that by adding a large cost M to the cost of each inter-cluster edge this problem can be reduced to a TSP. More details about this problem can be found in [61].

Generalized TSP: As in the case of clustered TSP, let $V_1, V_2, \ldots, V_k$ be a partition of the set of vertices of graph G. In a Generalizes TSP (GTSP), the goal is to find a shortest cycle in graph G which goes through exactly one vertex from each cluster $V_i, 1 \leq i \leq k$. Note also that if $|V_i| = 1$ for all i, GTSP is the same as TSP. The GTSP can be reduced to a TSP for arbitrary $|V_i|$ using the reduction described in [104]. WLOG, assuming that graph G is a digraph and the partitions are numbered in such a way that $|V_i| \geq 2$ for $1 \leq i \leq r$ and $|V_i| = 1$ for $r+1 \leq i \leq k$. For any $i \leq r$ let $V_i = \{i_1, i_2, ..., i_n\}$. For each arc $e \in E$, allow a new cost d_e defined as follows:

$$d_{i_j i_{j+1}} = -M, j = 1, \ldots, n_i \text{ with } n_i + 1 \equiv 1, i = 1, ..., r. \tag{2.5}$$

This guarantees that if a minimal TSP tour in graph G enters a cluster V_i through node i_j, it visits nodes of V_i in the order $i_j, i_{j+1}, ..., i_{n_i}, i_1, ..., i_{j-1}$ and leaves the cluster V_i from the vertex i_{j-1}. Now the goal is to interpret this outcome to be equivalent to a GTSP tour entering and leaving the cluster V_i by visiting vertex i_j. To demonstrate this fact, the new cost of arcs which are going out of i_{j-1} must coincide with the original cost of arcs leaving i_j. In the following Equation 2.6:

$$d_{i_{j-1}p} = c_{i_j p}, p \notin V_i, j = 1, 2, ..., n_i \text{ with index } 0 \equiv n_i, i = 1, ..., r \tag{2.6}$$

and $d_{uv} = c_{uv}$ for all other edges. From a minimal solution of the TSP on graph G with the modified costs d_e for $e \in E$, a minimal solution to GTSP can be recovered.

For more details on this problem, refer to [59, 75].

The MAX TSP: In contrast to the TSP, the objective in the MAX TSP is to find a tour in graph G where the total cost of edges of the tour is maximum. This problem can be solved as a TSP by just replacing each edge cost by its additive inverse. If problem requires that the edge costs are non-negative, a large constant could be added to each of the edge-costs. These replacing will not change the optimal solutions of the problem. The MAX TSP has been discussed in [60].

Traveling Tourist Problem: A tourist wishes to see all monuments in a city, and so must visit each monument or a neighbor thereof. Furthermore, it is assumed that a monument is visible from any of its neighbors. The edges therefore represent lines of a sight. The resulting walk will therefore visit a subset of all nodes in the graph G. For the Traveling Tourist Problem we refer to the [90].

The bottleneck TSP: In this variation of TSP the objective is to find a tour in graph G such that the largest cost of edges in the tour is as small as possible. A bottleneck TSP can be described as a TSP with exponentially large edge costs. For more details, refer to [60].

TSP with multiple visits (TSPM): In this problem the objective is to find a routing of a traveling salesman. Salesman starts at a given vertex of graph G, visits each vertex at least once and comes back to the starting vertex in such a way that the total distance traveled is minimized. The TSPM can be transformed into a TSP by replacing the edge costs with the shortest path distances in graph G. In the lack of negative cycles, shortest path distances between all pairs of vertices of a graph G can be computed using efficient algorithms [1]. If graph G contains a negative cycle, then TSPM is unbounded. More details on TSPM can be found in [60].

2.1.5 All-Pairs Shortest Paths

In this section, the problem of finding shortest paths between all pairs of vertices of graph is considered. Given is a weighted, directed graph $G = (V, E)$ with a weight function $\omega : E \rightarrow \mathbb{R}$ that maps edges to real valued weights. The objective is to find, for every pair of vertices $u, v \in V$, the shortest path from u to v. In addition the weight of a path is the sum of the weights of its edges. The all-pairs shortest-paths problem, can be solved, by running a single-source shortest-paths algorithm $|V|$ times, once for each node as the source. In some particular case, where all edge weights are non-negative, *Dijkstra's algorithm* [27] can be used for solving the problem. If the linear-array implementation of the min-priority queue is used, then the running time is $O(V^3 + VE) = O(V^3)$. Furthermore, the binary min-heap implementation of the min-priority queue generates a running time of $O(VE\ lg\ V)$, which is an improvement, if the graph is *sparse*. If the graph contains the negative-weight edges, then Dijkstra's algorithm cannot be used. Alternatively, the *Bellman-Ford algorithm* [27] can be run, once from each node. The resulting running time is then $O(V^2E)$, which results to $O(V^4)$ on a *dense* graph. In addition, for solving the all-pairs shortest paths problem on sparse graphs the *Johnson's algorithm* [27] is used. For more details see [27]. In contrast to the single-source algorithms, which assume a distance-list representation of the graph, the *Floyd-Warshall algorithm* [27] uses a distance-matrix representation.

The Floyd-Warshall Algorithm

The Floyd-Warshall algorithm (FW) [27] is a simple and extensively used algorithm for computing the shortest paths between all pairs of vertices in an edge weighted directed graph $G = (V, E)$. The algorithm runs in $O(V^3)$. The Floyd-Warshall algorithm produces the correct result as long as no negative cycles exist in the input graph. The FW algorithm take into consideration the intermediate nodes of a shortest path, where an intermediate node of a simple path $p = \langle v_1, v_2, \ldots, v_l \rangle$ is any node of p other than v_1 or v_l, that is, any node in the set $\{v_2, v_3, \ldots, v_{l-1}\}$. The Floyd-Warshall algorithm depends on the following observation. During assumption that the nodes of G are $V = \{1, 2, \ldots, n\}$, consider a subset $\{1, 2, \ldots, k\}$ of nodes for some k. For any pair of nodes $i, j \in V$, consider all paths from i to j of which intermediate nodes are all drawn from $\{1, 2, \ldots, k\}$, and let p be a minimal-weighted path between them. The Floyd-Warshall algorithm takes advantage of a relationship between path p and shortest paths from i to j with all intermediate nodes in the set $\{1, 2, \ldots, k - 1\}$. Furthermore, this relationship depends on whether or not k is an intermediate node of path p.

Algorithm 1 Floyd-Warshall

```
1: procedure FLOYD-WARSHALL(V,W)
2:     for all $k \in V$ do
3:         for all $i \in V$ do
4:             for all $j \in V$ do
5:                 $W_{ij} := min(W_{ij}, W_{ik} + W_{kj})$
6:             end for
7:         end for
8:     end for
9: end procedure
```

The running time of the Floyd-Warshall algorithm shown in Algorithm 1 is appointed by the triple nested **for** loops of lines 2-4. Furthermore, each execution of line 5 takes $O(1)$ time, so the algorithm runs in time $O(V^3)$. Because of that the Floyd-Warshall algorithm is pretty efficient for even modest sized input graphs.

2.2 Algorithms for the TSP

In this section, different algorithms that have been recommended in the literature for the TSP will be examined. This section introduces the basic concepts of heuristic approaches for the TSP and give some of the theoretical results that have been examined in the literature. Heuristic approaches correspond to approximation algorithms with objective to find the near optimal solutions quickly rather than the best solution to a given problem. The scope of the section mainly concentrates on the classical constructive, local search approaches and by nature inspired techniques and their extensions. For the TSP these diverse approaches represent the state-of-the-art approaches when the objective is to find satisfactory solutions quickly. Heuristic approaches for the TSP have been presented in several surveys. For the comprehensive overview of the heuristic techniques, refer to the surveys [70, 74] and to a book on the TSP and its variants [60].

2.2.1 Tour Quality

The classic measure of the quality of a tour length is the gap to the optimal tour length. This gap is commonly demonstrated as the pass over optimum. For algorithm A this gap can be formally shown as relative difference between obtained length of a tour l_A and optimal length of a tour l_C:

$$\frac{l_A - l_C}{l_C}. \tag{2.7}$$

Luckily, the standard testbeds, such as Reinelt's TSPLIB [116], provides a wide set of instances, which also includes the optimal tour length for most of tested instances. The optimal tour lengths are usually unknown for instances of sizes or structure that cannot be solved to optimality in a reasonable amount of time. Because of the lack of the optimal tour length for instances, a different reference point is needed. Those reference points present values, which are close to the optimal tour length. Furthermore, this reference points are as a consequence commonly a lower bound on the optimal solution.

The Held-Karp lower bound (HKLB) [65] represents a standard lower bound for the TSP. The HKLB on the optimal solution corresponds to the solution of a linear programming relaxation [22] of the standard integer programming formulation of the TSP. This implies that the integer constraints of the integer programming problem are substituted with bounded variables. Furthermore, the resulting linear programming problem [126] is then solved to optimality by using an algorithm such as the simplex method [31]. Even if the resulting linear program consists of an exponential number of sub-tour constraints it can be solved in polynomial time, because there exists a polynomial time "separation oracle" for the sub-tour constraints based on calls to the max-flow algorithm [76]. Experimental results, which are presented in [60], show that this lower bound is very close to the optimal tour length in most cases.

2.2.2 Tour Construction Algorithms

The objective of tour construction approaches is to build a tour for an instance of TSP from scratch. This can be achieved by constructing a tour which will follow some construction rule. Determinations during the construction of a tour are mostly made in a greedy fashion. The objective of these constructive methods is an immediate gain instead of looking ahead. There are many constructive heuristics which have been suggested for the TSP. A comprehensive overview of tour construction approaches for the TSP can be found in [10, 117]. Furthermore, for the thorough performance testing of the tour construction approaches for the symmetric TSP, especially for the Euclidean TSP, see [60, 117].

The tour construction approaches described in the next two subsections were chosen because of their suitability to symmetric, especially Euclidean instances, of TSP. Two basic and from the dissertation point of view important tour construction approaches are the *Nearest Neighbor* and the *Insertion* heuristics.

Nearest Neighbor Heuristic

One of the most intuitive heuristic algorithm for the TSP is the Nearest Neighbor algorithm (NN). The salesman starts at an arbitrary selected node and then sequentially moves to the nearest node he has not yet visited. When all nodes have been visited, salesman returns to the node he started from. The pseudocode of this algorithm for a set of nodes, i.e., cities $C = \{c_1, c_2, ..., c_n\}$ is shown in Algorithm 2.

The NN algorithm has a computational complexity of $O(n^2)$ which can be reduced to $O(n \log n)$ for geometric instances [10]. The best performance ratio noted for the instances of TSP, which satisfy the triangle inequality is given by $\frac{NN(i)}{OPT(i)} \leq 0.5 \left(\lfloor \log_2 n \rfloor + 1\right)$, see [70], and therefore growing in n. Nevertheless, in another case where the triangle inequality is not satisfied one can anticipate even worse the performance ratio.

In the literature exists various variants of the NN approach with objective to improve its performance. One such variant is the Double Ended Nearest Neighbor (DENN) [117], where the tour is built from both ends of the current sequence. Another variant is so-called Random Nearest Neighbor (RNN), where the node, which will be visited next is selected randomly from a set of nearest neighbors [60].

Algorithm 2 Nearest Neighbor

```
1: procedure NearestNeighbor(C)
2:     Select arbitrary city $c_j$, set $k = j$ and $C = \{c_1, ...c_n\} \setminus c_j$
3:     while $(C \neq 0)$ do
4:         Determine $c_l \in C$ with $d\,(c_k, c_l) = \min_{c_j \in C}(d(c_k, c_j))$
5:         Add $c_l$ to the tour by connecting $c_k$ to $c_l$ and set $c \leftarrow C \setminus c_l$ and $k = l$
6:     end while
7:     Connect $c_k$ to starting city $c_j$
8: end procedure
```

Insertion Heuristic

For the insertion heuristics a different construction rule is followed. The algorithm start with a sub-tour, which contains one or two cities. Then it successively add cities to the current sub-tour. This adding is followed by some selection criteria. The pseudocode of this approach is presented in Algorithm 3.

From the description of the algorithm three questions arise: From which city or cities to start the heuristic? Which city to insert next? Where to insert the city into the current sub-tour? Various decision rules have been proposed in the literature

Algorithm 3 Insertion

```
1: procedure INSERTION
2:     Select a starting tour through $k$ cities $T = \{c_1, c_2, \dots, c_k\}$
3:     repeat
4:         Select a city $c_i$ with $c_i \notin T$ following some criteria
5:         Insert city $c_i$ into the current sub-tour $T$
6:     until $c_i \notin T, i = 1, 2, ..., n$
7: end procedure
```

regarding the answers on these questions. The set of cities, which construct the starting sub-tour is commonly selected randomly and consists of one, two or three cities. For Euclidean problems other approaches such as starting the heuristic from the convex hull of the problem have been also suggested [117]. The last two questions, to which city to insert next and where to insert it, are tightly related. Some selection rules are:
1. Random Insertion: Choose the city to be inserted randomly from the set of cities that are not yet a component of the tour,
2. Cheapest Insertion: Insert the city, which is not yet a component of the tour, whose insertion results to the minimal grow in tour length,
3. Farthest Insertion: Insert the city, which is not yet a component of the tour, whose minimal distance to a city, is maximal,
4. Nearest Insertion: Insert the city which is not yet component of the tour and is nearest to any city which is at present a component of the tour. Note that previous section provides an idea for construction of nearest insertion.

A comprehensive empirical analysis of Insertion Heuristics with different insertion criteria can be found in [117]. Nearest insertion, farthest insertion and random insertion can be implemented to run in $O(n^2)$ time. Cheapest insertion has time complexity $O(n^2 \log n)$ and is computationally more expensive.

2.2.3 Local Search Algorithms

Local search algorithms for the TSP are built on simple tour adjustments. A local search algorithm is built from operations called moves, which are used to transform one tour to another. The local search is actually a neighborhood search process, where each tour has an alike neighborhood of tours. The local search algorithm repeatedly moves to a better neighbor as far as no better neighbors exist. These moves, which have been proposed for the TSP, can be sorted into operators of edge exchange, node exchange and node insertion. The edge exchange operators exchange the edges in the tour and will be described in detail in later section. The node exchange operator work in such a way that it exchanges two nodes in the sequence. On the other hand the node insertion operators work by deleting a node from a tour and inserting it at another position in the tour.

A tour t^* of the TSP example is called locally optimal, when all other tours in its neighborhood are at least as long as t^*. To identify a local optimum, the neighborhood of an initial tour is explored for a solution of better quality. When a new best solution s' has been found, it is accepted and its neighborhood $N(s')$ is searched. Operations of this kind are repeated as far as no more improvements can be found. A pseudocode using above notifications of solutions and neighborhoods is given in Algorithm 4.

Algorithm 4 Local Search

```
1: procedure LocalSearch
2:     Generate feasible Solution $s$
3:     Select a neighborhood function $N$
4:     repeat
5:         Search neighborhood $N(s)$
6:         if ($s'$ with $f(s') < f(s)$ found) then
7:             Set $s = s'$
8:         end if
9:     until no feasible lower cost solution $s'$ is found
10: end procedure
```

k-opt Local Search

The most studied algorithms of all of the local search algorithms are k-exchange neighborhood, which is an example of edge exchange algorithm. The k-exchange neighborhood is for the TSP commonly referred to as k-opt. For the TSP this local search algorithm can be defined as follows. Let S corresponds to the set of all tours of a TSP instance. Let us also introduce a metric p between tours in S, which measures the distance between tours with the number of edges not mutual to both, i.e., the number of edges, which represent part of tour T, but not part of tour T'. The k-opt local search can then be stated by:

$$N_k(T) = \left\{T' | p(T, T') = k, T' \in S\right\}.$$

From the above definition we should see that this statement introduces a whole set of search neighborhoods which is parameterized by k. Usually, the higher the k of a k-opt local search, the better the resulting tours. In spite of this, since the neighborhood size grows exponentially with k, only small k appear to be practical.

2-opt

The 2-opt local search illustrates the simplest of the k-opt local search algorithms for the TSP. The 2-opt algorithm was first proposed in [29], even though the basic move had already been suggested in [48]. This move deletes two edges, as a consequence the tour is aparted into two segments, and then algorithm reconnects those segments in the other possible way. An illustration of this edge removal and reconnection operation is presented in Figure 2.1.

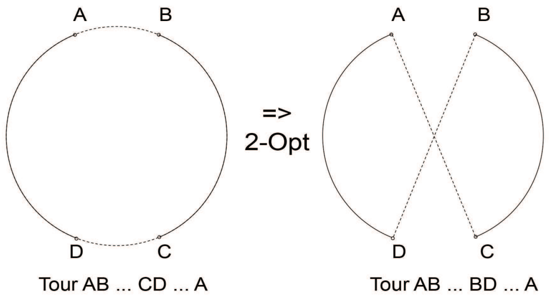

Figure 2.1: Edge removal and reconnection of 2-opt algorithm

Just as is shown in the figure, this approach is equal to a tour segment reversal. The order of the tour segment $B, ..., C$ is reversed with respect to $C, ..., B$. For the Euclidean TSP, 2-opt local search removes the *crossings* of edges in the tour. For symmetric TSPs a 2-opt move usually produce an improvement to the current tour if $d(A, C) + d(B, D) < d(A, B) + d(C, D)$.

There are two different representations of a 2-opt move, i.e. the move presented in the Figure 2.1 is equivalent to the move, where the tour segment between D and A is reversed. To denote an improvement, for a 2-opt move, either $d(A, B) > d(B, D)$ or $d(C, D) > d(A, C)$ or both must hold. Because of this the attention can be concentrated on moves satisfying $d(A, B) > d(B, D)$. This implies that once A and B are fixed we can bound the search for an improving move on cities D that are closer to B than A.

So to take advantage of this feature Steiglitz and Weiner [122] introduce a data structure to specify suitable candidates quickly. And so, storing for each city a list of the remaining cities sorted with increasing distances. When seeking for a 2-opt move, one has to start at the beginning of the list of B, move forward through the list observing its members until $d(A, B) \leq d(B, D)$. The difficulty of this approach is that creating the lists has time complexity $\Theta(n^2 \log n)$ and requires space quadratic in n. As a consequence in practice this list is commonly limited to only k-nearest neighbors for each city with fixed k, even if the resulting tour may not be locally 2-optimal. In practice only a small loss of tour quality is reported [70].

The Lin-Kernighan Algorithm

For over 30 years the world's champion heuristic for the TSP was usually recognized to be the local search algorithm of Lin and Kernighan (LK) [91]. LK algorithm is both an abstraction of the k-opt local search algorithm and an upgrade of perceptions the same authors had formerly used to the problem of graph partitioning. The fundamental idea of the LK heuristic is to develop composite moves by combining simple sub-moves to replace a variable number of edges. Because of this property this technique is also called a variable depth k-opt. The sub-moves commonly employed are simple k-opt local search moves, usually 2-opt and 3-opt moves.

The LK algorithm can be described as follows. In each step, the tour is broken up at one node forming a spanning tree with an extra edge (1-tree). By breaking up one edge of the degree-3 node and connecting the two degree-1 nodes this 1-tree can be transformed into a possible TSP tour. Consequently, the algorithm performs successive changes of edges until no more swappings are possible or until the best k-change in an iteration is found. A more comprehensive description of the LK algorithm can be found in the paper by Lin and Kernighan [91]. The original Lin-Kernighan heuristic was suggested for symmetric TSPs only.

Besides the high effort needed for its implementation the main drawback of the LK algorithm is its rather long running time. Because of this, several improvements to the genuine algorithm have been made. In the paper [117] Reinelt describes a variation of LK algorithm using the segment reversals and node insertion. In this variant, a special 3-opt move consists of a single node to enlarge the Lin-Kernighan neighborhood. Some different approaches can be found in [132] and in [114]. In these two papers the search neighborhood is the so-called flower transition, firstly suggested by Glover in [54] where the base is a 1-tree consisting of a cycle and a path attached to it.

An effective variation of the LK algorithm based on a sequence of 5-opt moves instead of 2-opt segment reversals was suggested in 2000 by Helsgaun [66]. The estimation of these move sequences is accomplished by considering very small candidate lists. Conversely, this candidate lists are determined in a very complex manner. Other approaches and variants of the LK algorithm have try to beat problems produced by very large TSP instances. One variant of the LK algorithm, which suggests an improvement for LK searches was presented in [6]. For a comparison of different Lin-Kernighan implementations we refer to a Johnson and McGeoch work on heuristics for the symmetric TSP [71].

2.2.4 Nature-Inspired Algorithms

The area of nature-inspired computing has grown in popularity over the last fifty years. Many of the nature-inspired algorithms currently in use are being applied to a wide range of problems, among them are of course ones tackling the TSP. The term *nature* is used to refer to any component of the universe, which is not a product of planned human design. The nature-inspired algorithms are in the category of meta-heuristic algorithms, where little or no problem particular information is used in the design of the algorithm. If we restrict our attention to the biological part of nature, we can highlight some of the useful properties. In nature, managing the trade-off between solution quality and time is basic to survival. A likewise trade-off is useful when using a heuristic to solve the optimization problems.

The fittest [1]individuals are those with supreme problem-solving feature. It is these problem-solving features which have been the source of inspiration for many nature-inspired techniques. We will now represent several meta-heuristic examples of nature-inspired algorithms for optimization problems, with focus on those techniques, which is used for tackling the TSP.

[1]The fittest individuals here means the best individuals. This kind of glossary is used respecting the vocabulary of the natural scientists.

Simulated Annealing

Simulated Annealing (SA) was introduced by Kirkpatric et al. in [81]. The SA optimization method builds on a similarity derived from physics processes, where a low energy state of a solid is inquired by an annealing procedure. The similarity with combinatorial optimization occurs, when the optimal solution to a given combinatorial optimization problem corresponds to the lowest energy of the solid. Therefore, a solution of a problem is perturbed and a neighbor solution is established with probability according to a Boltzmann distribution $e^{\frac{-\Delta E}{k*T}}$. Here ΔE correspond to the difference in quality between the current solution and the perturbed one. Furthermore, k is a scaling parameter and T correspond to the temperature of the process.

The higher the temperature the greater is the probability of acquiring a perturbed solution. When the temperature is low, improving moves will be privileged. By reducing T using an "annealing schedule" it is possible to simulate the freezing process. The pseudocode of the SA is shown in Algorithm 5:

Algorithm 5 Simulated Annealing

```
 1: procedure SimulatedAnnealing
 2:     $t = T(0)$, $n = 1$
 3:     best solution $s_{best} = S$
 4:     while (Termination Criterion Unfulfilled) do
 5:         Generate $s' \in N(s)$
 6:         /* we obtain a neighbor of $s$ applying the move operator $N()$ */
 7:         $\Delta f = f(s) - f(s')$
 8:         /* we calculate the difference in fitness */
 9:         if $((\Delta f \leq 0)$ or $(e^{\frac{-\Delta E}{k*T}} > random\,[0,1]))$ then
10:             $s = s'$
11:             /* we acquire the perturbed move if it is better than the current */
12:             /* or if the Boltzmann criterion is fulfilled */
13:         end if
14:         if $(f(s) > f(s_{best}))$ then
15:             $s_{best} = s$
16:         end if
17:         $t = T(n)$
18:         /* we use the annealing schedule for time $n$ */
19:         $n = n + 1$
20:     end while
21:     Return $s_{best}$
22: end procedure
```

Note the comments in pseudocode, which are there for better comprehension of an algorithm. The efficiency of the SA in solving a specific combinatorial optimization problem stands in the definition of the the annealing schedule and move operator. Since the first appearance of SA, the TSP has served as a test problem, considered in the original paper of Kirkpatrick [81]. For this particular approach the SA neighborhood was a 2-opt local searcher. In addition, Boese's PhD dissertation [13] comprehensively analyzes the optimal annealing schemes for the TSP.

Ant Colony Optimization

A recent and more and more popular meta-heuristic technique inspired by nature is the Ant Colony Optimization (ACO) introduced in 1997 by Dorigo [38]. ACO presents a population based approach, that was inspired by the behavior of real ants in nature. The essence of this technique is to exploit the fact that ants in nature appear very effective in finding the shortest paths to a source of food. Ants are able to find the shortest paths through communication transmitted by leaving pheromone trails while exploring for food. For the TSP, ACO represents a constructive technique, which is frequently updated until some stopping condition is reached. Starting with n ants, each positioned at a different node, the ants consecutively move along the edges to create feasible tours. Determination, on which node to visit next are made in a probabilistic manner established on the pheromone trail left at earlier investigation. The pheromone trail is updated after each iteration depositing a higher amount of pheromone at edges which was used in the shortest tours.

There exist several improvements to this basic approach that have been proposed in the literature. Especially the conjunction with local search algorithms to speed up the search process [123]. The computational results described in the literature range from very poor to moderate quality in [39] to solutions of very good quality presented in [123]. More information on ACO for the TSP can be found in [39, 123].

Evolutionary Algorithms

Evolutionary Algorithms (EA) simulate the process of biological evolution in nature. EAs are search methods which are motivated by natural selection and survival of the fittest from biological world. EA performs a search within a population of solutions. Each iteration i.e. generation of an EA includes a competitive selection among all solutions in the population. This selection results in survival of the fittest and erasure of the poor solutions from the population. Recombination is performed by exchanging parts of a solution with another one. This process forms the new solution that may be better than the previous ones. Furthermore, a solution can be mutated by manipulating a part of it. The evolutionary operators, recombination i.e. crossover and mutation, are used to develop the population to areas of the search space, in which good solutions exist. Four main evolutionary algorithm classes have been introduced: genetic programming is a computational method, which was proposed by Koza [82], Evolutionary Strategies (ES) introduced by Rechenberg [112], Evolutionary Programming (EP) developed by Fogel [49], and Genetic Algorithms (GA), proposed by Holland [67]. The general template of an EA is shown in Algorithm 6. We should note that the P in pseudocode stands for the population of the individuals. Note also, that all mentioned variants of evolutionary algorithm (GA, EP, and ES) are special cases of this scheme.

Algorithm 6 Evolutionary Algorithms

```
1: procedure EA
2:     $t := 0$
3:     initialize population ($P(0)$)
4:     evaluate ($P(0)$)
5:     repeat
6:         $P' :=$ select for variation ($P(t)$)
7:         recombine ($P'$)
8:         mutate ($P'$)
9:         evaluate ($P'$)
10:        $P(t+1) :=$ select for survival ($P(t), P'$)
11:        $t := t+1$
12:    until terminate = true
13: end procedure
```

For the TSP the EAs may be shortly outlined as follows. Starting from a population of t individuals i.e. tours, select t' different individuals for mutation and parents for mating. Perform mutations on the selected individuals and the mating recombination. In recombination the information of two parent individuals are combined in order to create an offspring individual. Then select t existing individuals of the new population succeeding the selection strategy. This procedure is consecutively applied for some iterations, i.e., generations until some termination criteria is met. For further reading on EA we refer to [7, 95, 100, 121].

Genetic Algorithms

Genetic Algorithms (GA) were introduced by Holland in the 1970s [67]. These methods are adaptive search techniques based on the instrument of natural selection and the survival of the fittest concept. A comprehensive introduction to GA is given in Goldberg's book [56]. The main inspiration behind GA is to start with randomly created initial solutions and implement the survival of the fittest strategy to develop the better solutions through iterations i.e. generations.

The pseudocode of a GA can be seen in Algorithm 7.

Same as in the case of EAs the P in pseudocode stands for the population of the individuals. Furthermore, a GA process includes initial generation of individuals i.e populations (line 2 in the algorithm), evaluation of fitness (line 4 in the algorithm), selection of chromosomes (line 5 in the algorithm) and applying the genetic operators for reproduction, recombination and mutation (line 6 in the algorithm). How to encode a search solution is a basic and key issue in designing a Genetic Algorithm [21]. Many optimization operators for TSP were proposed by Goldberg [56]. A usually used strategy for encoding is a transposition expression [113]. In the transposition expression encoding strategy, each city of the TSP is encoded as a gene

Algorithm 7 Genetic Algorithm

```
1:  procedure GA
2:      Randomly generate an initial population $P(t)$
3:      while (Termination Criterion Unfulfilled) do
4:          Compute the fitness $f(p)$ $\forall p \in P(t)$
5:          According to $f(p)$ choose a subset of $P(T)$, store them in $M(t)$
6:          Recombine and mutate individuals in $M(t)$, store results in $M'(t)$
7:          Generate $P(t+1)$ by selecting some individuals from $P(t)$ and $M'(t)$
8:          $t = t + 1$
9:      end while
10:     Return best $p \in P(t-1)$
11: end procedure
```

of the chromosome. This encoding includes the constraints that each city appears only once in the chromosome. Transposition expression is one of the best expression for TSP which based on the order of tour, on the other hand such a procedure may leads to infeasible tour after traditional recombination operator. This is a usual case in TSP. Even if feasibility can be maintained in numerous ways by some repair algorithms, such methods can spend a substantial amount of time and oftenly retain convergence [113].

To overcome a possible creation of infeasible tours another encoding method was introduced. It is the Random Keys encoding [9], which was introduced by Bean. A random numbers are used to encode the construction of the solution in random keys encoding. Such representation guarantees that feasible tours are preserved during the use of genetic operators.

In the GA, the recombination and mutation are two most important factors for the success of the algorithm. Recombination is a crossover operator that takes two individuals i.e genomes, and cuts their chromosome strings at a number of chosen positions. The sub-segments produced by that cuts are then combined and reconnected to produce an entire chromosome with characteristics of the two parents. Therefore, the basic role of recombination is information exchange between successful solutions.

Numbers of different GA recombination operators have been proposed in the literature to solve the TSP. The linear order crossover [33], partially mapped crossover [56] and order based crossover [9,56] are the usually used recombination techniques for TSP. Aside from this three usually used recombination algorithms, many different recombination operators are proposed for the TSP, for example: edge map crossover [50], distance preserving crossover [97], sub-tour crossover [131], generic crossover [98], edge assembly crossover [101], natural crossover [73], greedy search crossover [102], heuristic based crossover [89].

The other GA operator usually used is mutation. It is used to produce the variation of genomes. Mutation is applied stochastically to a child after recombination. It alters one or more genes with a small probability granting a small amount of random search. As a result of that, no point in the whole search space has a zero probability of being inspected by the genetic algorithm. Therefore, a mutation operator is used to enhance the diversity and provide a chance to escape from local optima.

Many mutation operators were proposed in the literature. Some of them are: insert, inverse, swap, displace, hybrid mutation [78], and heuristic mutation. The first five listed here are realized by small modification of genes. Heuristic mutation was proposed by Cheng and Gen [20]. This operator adopts a neighborhood strategy to improve the solution.

Overall the GAs have shown to be useful and efficient when the search space is large, complex or poorly understood. A lot of progress has been made recently. In 2006, Carter and Ragsdale propose a new GA chromosome and related operators for the Multiple TSP [19]. In 2007, Nguyen described a hybrid GA based on a parallel implementation of a multi population steady-state GA involving local search heuristics [103]. For further reading on GA we refer to [128, 131, 134]

In next two subsections we show two recombination operators, Edge Map Crossover and Distance Preserving Crossover.

Edge Map Crossover

Edge Map Crossover (EMX) [50] is an implementation of the recombination operator for GA. It makes use of a so called edge map. Edge map is a table in which each location is placed. For each location there is a list in which the neighboring locations are registered with this location. Recombination is then built as follows. Select the first location of one of both parents to be the current location. Second step is to delete the current location from the edge map lists. If the current location still has remaining edges, go to the previous step, otherwise go to the next step. Select the new current location from the edge map lists of the current location as the one with the shortest edge map list. If there are remaining locations, select the one with the shortest edge map list to be the current location and return to second step. An example of EMX operator is scripted below in section. We should note that an explanation of the example follows a procedure composed above.

```
Example:
Parents: 1-2-3-4-5-6; 2-4-3-1-5-6
Edge map: 1) 2 6 3 5;  2) 1 3 4 6;  3) 2 4 1;
          4) 3 5 2;    5) 4 6 1;    6) 1 5 2 6
1. Random choice: 2,
2. Next candidates: 1 3 4 6, choose from 3 4 6 same#edges, choose 3,
3. Next candidates: 1 4 (edge list 4 < edge list 1), choose 4,
4. Next candidate:  5, choose 5,
5. Next candidate:  1 6 (tie breaking) choose 1,
6. Next candidate;  6, choose 6.
Offspring: 2-3-4-5-1-6
```

Distance Preserving Crossover

Distance Preserving Crossover (DPX) is another implementation of the recombination operator for GAs. DPX attempts to create a new tour with the same distance to both parents [66]. In order to establish this, the content of the first parent is copied to the offspring i.e child and all edges, that do not occur in the second parent, are removed. The resulting fragments are reconnected without making use of non-overlapping edges of the parents. If edge (i, j) has been destroyed, the nearest available neighbor k of i from the remaining fragments, is selected and the edge (i, k) is added to the tour. An example of DPX operator is scripted below.

```
Example:  Parents: 5-3-9-1-2-8-0-6-7-4; 1-2-5-3-9-4-8-6-0-7
Fragments: 5-3-9|1-2|8|0-6|7|4
Offspring: 6-0-5-3-9-8-7-2-1-4
```

We procede to next subsection which in more details describes the hybrid genetic algorithms.

Hybrid Genetic Algorithms - Memetic Algorithms

Hybrid Genetic Algorithms are evolutionary algorithms that interpolate a phase of particular optimization or learning as part of their search procedure. Earliest references on Hybrid Genetic Algorithms are tracked to [47,64,127]. The most basic Hybrid Genetic Algorithm can be seen below in Algorithm 8 :

Algorithm 8 Hybrid Genetic Algorithm

```
1: procedure HybridGA
2:     Randomly generate an initial population $P(t)$
3:     while (Termination Criterion Unfulfilled) do
4:         Compute the fitness $f(p)$ $\forall p \in P(t)$
5:         According to $f(p)$ choose a subset of $P(T)$, store them in $M(t)$
6:         Recombine and mutate individuals in $M(t)$, store results in $M'(t)$
7:         Improve by local search $(M'(t))$
8:         Generate $P(t+1)$ by selecting some individuals from $P(t)$ and $M'(t)$
9:         $t = t + 1$
10:    end while
11:    Return best $p \in P(t-1)$
12: end procedure
```

Same as in the case of GAs the P in pseudocode stands for the population of the individuals. Furthermore, a process of hybrid genetic algorithms includes an initial generation of individuals i.e populations (line 2 in the algorithm), evaluation of fitness (line 4 in the algorithm), selection of chromosomes (line 5 in the algorithm) and applying the genetic operators for reproduction, recombination and mutation (line 6 in the algorithm). In the pseudocode above the difference from a standard i.e canonical GA is the use of local search. It is used to improve the newly created individuals.

We should note that this is just one possible way to hybridize a GA with local search. Even though it seems from the Algorithm 8, to be a naive minor change, is in fact a crucial deviance from a canonical GA. Hybrid genetic algorithms are inspired by example of adaptation in natural systems that combine evolutionary adaptation of populations of genomes with individual learning.

In the literature, Hybrid Genetic Algorithms have also been named Memetic Algorithms (MA) [59, 83–85, 98, 107], Genetic Local Searchers (GLS) [96], Lamarckian Genetic Algorithms [99], Baldwinian Genetic Algorithms [86]. The Memetic Algorithms differ from other hybrid evolutionary techniques that all individuals in the population are treated as local optimum, since after each mutation or recombination, a local search is applied. The name genetic local search (GLS) was firstly used by Ulder in [125] to describe an evolutionary algorithm with recombination and then applied local search. In [17], Bui suggest a GLS algorithm with Lin-Kernighan heuristic as the neighborhood procedure. They developed a k-point recombination operator with an additional repair mechanism for producing the feasible offspring. In [77] Katayama suggest an evolutionary algorithm with Lin-Kernighan algorithm and small populations of just two individuals and a heuristic recombination scheme.

2.2.5 Finding exact solutions of the TSP

Finding the exact solution to a TSP with n nodes involves to check $(n-1)!$ of possible tours. Evaluation of all possible tours is infeasible for even small instances of TSP. For finding the optimal tour Held and Karp [65] introduced the following dynamic programming formulation: Given a subset of node pointers, discarding the first node, $S \subset \{2, 3, ..., n\}$ and $l \in S$, let $d^*(S, l)$ stand for the length of the shortest path from node 1 to node l, visiting all nodes in S in between. For $S = \{l\}, d^*(S, l)$ is defined as d_{1l}. Then the shortest path for larger sets with $|S| > 1$ is:

$$d^*(S, l) = \min_{m \in S \setminus \{l\}} (d^*(S \setminus \{l\}, m) + d_{ml}). \tag{2.8}$$

In conclusion, the minimal tour length for a complete tour which involve bringing back to node 1 is:

$$d^{**} = \min_{l \in \{2,3,...,n\}} (d^*(\{2, 3, ..., n\}, l) + d_{l1}). \tag{2.9}$$

Using the Equation 2.8 and the Equation 2.9, the quantities $d^*(S, l)$ can be calculated recursively and the minimal tour length d^{**} can be obtained. In a next phase, the optimal permutation $\pi = \{1, i_2, i_3, ..., i_n\}$ of node pointers 1 through n can be calculated in oposite manner, starting with i_n and working consecutively back to i_2. This phase take advantage of the fact that a permutation π can be optimal only if

$$d^{**} = d^*(\{2, 3, ..., n\}, i_n) + d_{i_n 1} \tag{2.10}$$

and, for $2 \leq p \leq n-1$,

$$d^*(\{i_2, i_3, ..., i_p, i_{p+1}\}, i_{p+1}) = d^*(\{i_2, i_3, ..., i_p\}, i_p) + d_{i_p i_{p+1}} \tag{2.11}$$

The complexity of space for saving the values for all $d^*(S, l)$ is $(n-1)2^{n-2}$ which strictly restricts the dynamic programming algorithm to TSP of small sizes [62].

A quite different procedure can address larger instances by using a relaxation of the LP problem. This procedure iteratively constrain the relaxation until a solution is found. This technique for solving LP problems is called *cutting plane method* and was introduced by Dantzig, Fulkerson, and Johnson in 1954 [30].

Each iteration of a method begins with using instead of the original linear inequality description S the relaxation $Ax \leq b$. Here the polyhedron P determined by the relaxation contains S and is bounded. In addition the optimal solution x^* of the relaxed problem can be reached using standard LP solvers. If the x^* belongs to S, the optimal solution of the original problem is obtained. If not then a linear inequality can be found which is content by all points in S but disturbed by x^*. This inequality is called a cutting plane or cut. If no additional cutting planes can be found or the improvement becomes very small, the problem is branched into two sub-problems. These sub-problems can be minimized individually. Branching is done step by step that leads to a binary tree of subproblems. Then each of a subproblem is solved without further branching or is found to be irrelevant. Irrelevant means, that relaxed version previously produces a longer path than a solution of another subproblem. This method is called branch and cut and was introduced by Padberg and Rinaldi, in 1990 [110]. The branch and cut method is a variation of the branch and bound procedure presented by Land and Doig, in 1960 [87].

The initial polyhedron P used by Dantzig [30] includes all vectors x such that for all $e \in E$ is $0 \leq x_e \leq 1$. Furthermore, in the resulting tour each node is connected to exactly two other nodes. Different methods for finding cuts to prevent sub-tours (*sub-tour elimination inequalities*) and to guarantee an integer solution, *Gomory cuts*, were built over time [62]. At present the most effective accomplishment of this technique is *Concorde* described in [4]. Concorde is a computer code for the symmetric TSP. The code is written in the *AnsiC* programming language. At the time of writing this dissertation the Concorde's TSP solver has been used to obtain the optimal solutions to 106 of the 110 instances from TSPLIB [116]. In Chapter 3 the Concorde's TSP solver was used for computing the lower bound for the quality of solutions for tested algorithms. Furthermore, in Chapter 4 it was used as a solver for the TSP.

Thus we conclude Chapter 2. In Chapter 3 we systematically study approaches to Hybrid Genetic Algorithm. Furthermore, we investigate some of the constructive, local search and by nature inspired approaches that are described in this chapter and compare their performance to proposed grafted genetic algorithm for solving symmetric TSP.

Chapter 3

Grafted Genetic Algorithm for Traveling Salesman Problem

3.1 Introduction

Genetic Algorithms (GA), which were described in detail in Section 2.2.4 use some mechanisms inspired by biological evolution [67]. They are applied on a finite set of individuals called population. Each individual in a population represents one of the feasible solutions in the search space. Mapping between genetic codes and the search space is called encoding and can be binary or over some alphabet of a higher cardinality. Good choice of encoding is a basic condition for successful application of a genetic algorithm. Each individual in the population is assigned a value called fitness. Fitness represents a relative indicator of quality of an individual compared to other individuals in the population. Selection operator chooses individuals from the current population and takes the ones, that are transferred to the next generation. Thereby, individuals with better fitness are more likely to survive in the population‘s next generation. The recombination operator combines parts of genetic code of the individuals (parents) into codes of new individuals (offsprings). Such a mixing of genetic material enables that well-fitted individuals or their relatively good genes give even better offspring. By a successive application of selection and crossover, the diversity of genetic material can be decreased, which leads to a premature convergence in a local optimum, which may be far from a global one.

The components of the genetic algorithm software system are: Genotype, Fitness function, Recombinator, Selector, Mater, Replacer, Terminator, and in our system a (Local) Optimizer which is a new extended component. In the Traveling Salesman Problem (TSP) a set $\{c_1, c_2, ...c_n\}$ of cities is considered and for each pair (c_i, c_j) where $i \neq j$, a distance $d(c_i, c_j)$ is given. The goal is to find a permutation π of the cities that minimizes the quantity

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}). \tag{3.1}$$

This quantity is referred to as the tour length since it is the length of the tour a salesman would have to travel when visiting the cities in the order specified by the permutation π, returning at the end to the initial city. We will concentrate in this chapter on the symmetric TSP, in which the distances satisfy $d(c_i, c_j) = d(c_j, c_i)$ for $1 \leq i, j \leq n$ and the distance is Euclidean. The TSP is known to be *NP-hard* [53], even under substantial restrictions, we are restricted to compute approximate solutions. Details about TSP can be found in Section 2.1.

There were proposed various greedy heuristics for solving TSP including nearest neighbor, details can be found in Section 2.2.2 and 2-opt. We will use the later in our algorithm as a hybridization component of canonical GA. The 2-opt is a simple optimization algorithm for the TSP, it is in detail described in Section 2.2.3. The main idea is to take a route that crosses itself and reorder it, so that it does not cross itself any more by reducing the tour length. An exchange step consists of removing two edges from the current tour and reconnecting the resulting two paths in the best possible way, as shown in Figure 2.1. Once we choose the two edges to delete, we do not have a choice about which edges to add. There is only one way to add new edges, that results in a valid tour.

The 2-opt optimization is used to hybridize GA meta-heuristic to solve TSP. Although the 2-opt algorithm [44, 66] performs well and can be applied to TSP with many cities, it finds only a local minimum. Furthermore, for the general (i.e. non-Euclidean) version of TSP it is known that there is no upper bound on a quality of a heuristic algorithms unless $P = NP$ [53]. Nevertheless, we will be interested in relative quality of our algorithm on a given set of samples.

In practice one of the most frequently used programs for solving TSP is Concorde [4]. It is based on the branch and cut method [62]. Besides it there is a range of solutions based on meta-heuristics including life-inspired approaches like ant colonies, see Section 2.2.4 and GA, see Section 2.2.4. Our algorithm is based on the latter. Although there are known approaches that tried to combine canonical GA with local optimization (cf. Memetic Algorithms in [98], or hybridization of GA in [119], details can be found in Section 2.2.4), our solution is novel in two respects. First, we include local optimization as one of the operators of GA and secondly we do not use it in all generations.

The rest of the chapter is organized as follows. First we introduce the general framework of our algorithm including the principle of methodology and present the procedure of hybridization of TSP solving genetic algorithm. The third section describes two experiments and instances of TSP used in the experiments. The section is followed by experimental results, their analysis and discussion. The chapter concludes by summarizing the results and conclusion.

3.2 Grafted GA for the TSP

Grafting in botany is when the tissues of one plant are affixed to the tissues of another. Grafting can reduce the time to flowering, shorten the breeding program, etc. Similarly we introduced into a canonical GA a local optimizer - we grafted GA or we hybridized it. This way we (locally) optimize genomes in an evolution process.

There exist a number of local optimizers, which can be used on their own as a greedy solution to NP-hard problems - e.g. Freisleben in [50] used a k-opt heuristics. There exists even its hardware implementation [68].

The pseudocode of our Grafted Genetic Algorithm (GGA) is listed in Algorithm 9.

Algorithm 9 Grafted Genetic Algorithm

```
1: **procedure** GGA
2:     $t = 0$
3:     $p(t) := Initialize()$
4:     $q(t) := Evaluate(P(t))$
5:     **while** $(q(t) < q_{expected})$ **and** $(t < t_{max})$ **do**
6:         $sel := Select(P(t))$
7:         $mat := Mate(sel)$
8:         $rec :=$ **for each** pair $m \in mat$ **do** $Recombine(m)$
9:         $loc :=$ **for each** genome $r \in rec$ **do** $Optimize(r)$
10:        $P(t+1) := Replace(loc, P(t))$
11:        $q(t+1) := Evaluate(P(t+1))$
12:        evaluate $(P(t+1))$
13:        $t := t+1$
14:    **end while**
15: **end procedure**
```

As usual, our algorithm stops either when the expected quality of solution is reached or when the maximum number of generations t_{max} has passed. The former can be measured in various terms starting from the absolute quality value to the relative diversification of population (e.g. standard deviation). On the other hand, the latter is measured either with the number of generations or with the total time elapsed. Tournament Selector (line 6 in the algorithm) places groups of genomes from the population together, creating the groups from top to bottom with respect to the enumerated ordering of the genomes in the population and selects the best of the genomes within this group. This is repeated until the required amount of genomes is selected. The Random Mater (line 7 in the algorithm) is a simple way of mating parents. It mates the parents as enumerated in the population at random using the mating size to create groups until no more groups can be created. The new offspring only replacer is the implementation of the classical replacement strategy that simply only allows the offspring to survive. Thus the genomes from the next generation replace the entire current population.

We studied two versions of recombination (line 8 in the algorithm). The first, *Edge Map Crossover, EMX* for short [98], uses a so-called edge map $EM[*]$ that contains a list of neighboring cities for each city. First the operator for each edge (u, v) in a parent genomes A and B, adds v to the edge-map list $EM[u]$ and u to $EM[v]$ for a symmetric version of TSP. Then the operator works as follows:

1. Pick a random city to be the current location u.

2. Remove the current location u from all edge map lists $EM[*]$.

3. If the current location $EM[u]$ still has remaining edges, go to step 4, otherwise go to step 5.

4. Choose the new current location u' from the edge map list $EM[u]$ as the one with the shortest edge map list $EM[u']$. Set $u := u'$ and go to step 2.

5. If there are left any locations, choose as the new current location u' the one with the shortest edge map list $EM[u']$. Set $u := u'$ and go to step 2.

The EMX is in more details presented in Section 2.2.4. The second recombination operator, that was studied, was a *Distance Preserving Crossover, DPX* for short [50]. It creates a new tour (offspring) preserving the same distance in the number of edges to both parents. In detail, the operator DPX creates an offspring C from parents A and B as follows:

1. Pick at random one of the parents ($wlog.A$) and copy it to C.

2. If $(u, v) \in C$ and $(u, v) \notin B$ delete it from C. At this point C contains fragments of connected cities $(u_l \to v_l), \dots, (u_k \to v_k)$, where in some fragments $u_i = v_i$. We call set of u_i and v_i the end-points of C, EP_c.

3. Pick at random a city x from EP_c and delete it from EP_c.

4. Pick from EP_c the closest city y to x so that there is edge (x, y) neither in A nor in B. Delete y from EP_c.

5. Merge fragments $(u_i \to y)$ and $(x \to v_j)$ into $(u_i \to v_j)$.

6. If EP_c is not empty, go to step 3.

The DPX is in more details described in Section 2.2.4. Although both recombination operators produced offsprings from valid parents, they produced them in a random way. Because of randomization, the selection and mating (lines 5 and 6 respectively) lost their role. Moreover, the randomization is a very good source of diversification. However, we are missing in our meta-heuristic the process of specialization i.e. intensification.

This was the reason to extend our algorithm with a specialization step - we *grafted* a canonical genetic algorithm with a local optimizer and obtained a *grafted genetic algorithm, GGA* [34–37]. We did not use a k-opt heuristics due to its complexity, but rather its simpler version 2-opt explained in previous section (see Figure 2.1). The hybridization occurs in line 9 of the pseudocode of Algorithm 9.

In [125], Ulder uses a 2-opt neighborhood structure for the local search in the improvement step, so the standard 2-opt heuristic is performed on each iteration of an algorithm. In our GGA we will not use hybridization in all generations. Furthermore, in Ulder's algorithm recombination is done by taking two tour at random and implanting a chosen sub-path of one of the tours, containing at most one third of all cities, into the other one. In our GGA we used a very different recombination operators, EMX and DPX.

3.3 Experiment

For testing our strategy and comparing it to other solutions we used the instances of symmetric TSP found on TSPLIB [116]. We used relatively small instances, for which best solutions are known. The goal of this research was not to find a better algorithm, but rather to study on a controlled environment the impact of grafting a genetic algorithm.

In the first experiment we used 20 instances, with different sizes in a range from 14 to 150 cities per instance. We studied our proposed method (grafted genetic algorithm (GGA)) using two different recombination operators: an edge map crossover (GGAemx) and a distance preserving crossover (GGAdpx). As the lower and upper limits on the quality of solution we used greedy heuristic and *Concorde* [4] respectively. For the sake of completeness we compared our method also with 2-opt heuristic itself and with a canonical genetic algorithm. The main difference between our method and canonical genetic algorithm is that we use local optimizer in every generation of the algorithm.

In the second experiment we studied what happens if we do not use local optimization in all generations – in evaluation it was used in 10, 20, 30, 40, 50, 60, 70, 80 and 90 percents of the generations. Furthermore, for each percentage we applied local optimization in three different ways: at random generations, at the initial generations and at the ending ones.

All experiments were conducted on a computer with Pentium(R) 2.8 GHz CPU and Windows 7 operating system. In our results we cannot directly compare the running times of different solutions as they were implemented in different programming languages. On one hand we used as a development environment for GGA the Java written *EA Visualizer* [15], while *Concorde* is an *AnsiC* application. However, we can compare running times of GGA for different instances and cases explained before.

3.4 Evaluation

We present results separately for the first and for the second experiment. In both experiments we used instances of TSP from a *TSPLIB* of various sizes. The name of the instance also contains its size (the first columns of Table 3.1). Next, in the experiments we measured three quantities: the wall clock time, the number of generations and the quality of the result. The latter was measured against the optimal solution obtained by Concorde. The quality of algorithm A is defined as

$$q_A = \frac{l_A - l_C}{l_C}, \tag{3.2}$$

where l_C is a path length obtained by Concorde and l_A is a path length obtained by A. We express the quality always in percents, where, for example, 4% means 4% worse than Concorde.

Let us look first at the results of the first experiment (cf. Table 3.1), in which we compared our GGA against greedy algorithm and Concorde. We also compared it against canonical genetic algorithm (GA). The termination condition in all genetic

Table 3.1: Five techniques for solving Euclidean TSP: greedy, 2-opt heuristic, GA with edge map crossover, GA with distance preserving crossover, grafted versions of the later and Concorde.

Name	**Greedy**	**2-opt**	**GAemx**			**GAdpx**			**GGAemx**			**GGAdpx**			**Concorde**	
	quality	*quality*	*quality*	*gen.*	*time*	*quality*	*gen.*	*time*	*qual.*	*gen.*	*time*	*qual.*	*gen.*	*time*	*opt*	*time*
burma14	8.32%	5.71%	0%	74	3.4	0%	81	3.5	0%	7	0.6	0%	6	0.5	3323	0.1
ulysses16	10.42%	7.15%	0%	136	4.1	0%	125	4.4	0%	9	0.7	0%	9	0.7	6859	0.2
ulysses22	12.54%	7.87%	0%	1267	14.7	0%	1328	16.4	0%	8	0.6	0%	8	0.7	7013	0.2
bayg29	13.37%	6.38%	0%	1345	19.4	0%	1137	17.6	0%	13	1.3	0%	14	1.4	1610	0.3
bays29	12.87%	5.37%	0%	2185	29.2	0%	2643	34.1	0%	12	1.2	0%	12	1.2	2020	0.3
dantzig42	14.06%	7.11%	0%	4704	79.8	0%	4232	74.6	0%	10	1.3	0%	9	1.3	699	0.5
att48	13.98%	8.47%	0%	4807	85.2	0%	5213	91.3	0%	22	2.2	0%	23	2.3	33522	0.6
eil51	15.24%	7.67%	4.21%	5482	100.0+	5.23%	5489	100.0+	0%	33	3.9	0%	30	3.8	426	0.3
berlin52	14.82%	7.45%	0%	2037	33.7	4.92%	5021	100.0+	0%	15	1.7	0%	15	1.7	7542	0.4
st70	13.17%	7.84%	5.12%	5259	100.0+	5.72%	5198	100.0+	0%	20	4.1	0%	19	4.1	675	0.5
eil76	14.47%	8.15%	6.56%	5347	100.0+	7.24%	5298	100.0+	0%	53	4.5	0.19%	49	4.4	538	1.3
pr76	13.96%	9.95%	4.18%	5218	100.0+	5.36%	5191	100.0+	0%	42	4.1	0%	43	4.2	108159	1.2
gr96	16.32%	7.14%	4.98%	5191	100.0+	5.71%	5090	100.0+	0%	73	8.4	0.13%	73	8.4	55209	1.6
rat99	14.79%	7.41%	5.31%	5114	100.0+	7.12%	5011	100.0+	0%	74	11.9	0.17%	70	11.7	1211	1.7
kroA100	12.37%	8.07%	5.12%	5072	100.0+	6.58%	4971	100.0+	0%	24	3.6	0.18%	22	3.5	21282	1.7
kroB100	16.58%	7.19%	6.14%	5041	100.0+	5.92%	4816	100.0+	0%	39	5.8	0.21%	36	5.7	22141	1.7
kroC100	10.47%	11.19%	4.87%	5121	100.0+	6.78%	4923	100.0+	0.10%	34	5.3	0.19%	28	5.1	20749	1.8
kroD100	14.81%	7.74%	5.07%	4976	100.0+	8.12%	4951	100.0+	0%	31	5.6	0.29%	25	5.3	21294	1.5
lin105	16.60%	9.85%	6.72%	4756	100.0+	6.51%	4803	100.0+	0.01%	26	4.6	0.17%	25	4.6	14379	1.3
ch150	19.62%	11.72%	7.22%	4512	100.0+	8.77%	4460	100.0+	0.22%	88	15.2	0.32%	86	15.1	6528	7

algorithms was: either standard deviation of genomes was 0 ((local) minimum was reached) or time 100 seconds time limit expired.

We first look at the quality of results, then at the running time and finally comment on a trade-off between the quality and the running time. The last column of Table 3.1 gives results of Concorde and actual path length in *opt* column. On the other side of the table is a greedy approach, which quality (column *quality* computed using Equation 3.2) is mostly in the range between 10% and 20%. However, application of a simple 2-opt heuristic on a randomly generated tour improves the quality to approximately 10% or even better. On the other hand use of GA further improves the quality to approximately 5%. Note, that runs in cases with 70 or more sites terminated due to time limit and hence minimum was not reached. All these results were expected.

The long running time of GA was a reason to graft (or hybridize) the GA with local optimization. The result was substantial decrease in running time. In all cases for GGAemx and for GGdpx the runs were terminated upon reaching the minimum. The reached minimum was, however, the local one. Nonetheless, we showed, that the combination of two methods improved the quality of results in a synergy. The quality of result was bellow 1% off the optimum.

The sixth column in Table 3.1 describes results obtained by GGAemx. In 17 out of 20 considered cases an optimal solution was found. Remaining three instances differ from optimal solution in 0.01, 0.10 and 0.22 percent. The solutions were found in relatively few generations and very fast. Execution times were 0.6 to 15.2 seconds.

The seventh column in Table 3.1 corresponds to results of GGAdpx. In 11 out of 20 considered cases an optimal solution was found. In remained 9 cases, delivered solutions differ from optimal in range from 0.13 to 0.32 percent. The running time and number of generations of GGAdpx, in comparison with GGAemx, are slightly lesser, particularly in the lowermost part of the table which represents more complex instances.

Quantitative results on the test cases from *TSPLIB* show that grafted algorithms, GGAemx and GGAdpx, have advantages. Even when their's components have serious drawbacks, their grafted combinations exhibits a very good behavior. Results on examples from *TSPLIB* show that this grafted method combines good qualities from both methods applied and outperforms each individual method.

The running times in Table 3.1 are given for all algorithms except greedy and simple 2-opt heuristics. The latter ones had running times in the range between 0.5 second and 1.5 seconds. However, since all algorithms except Concorde were programmed in Java, their running times are not directly comparable. Nonetheless, the relative increase in time as a function of a problem size can be compared, and this shows us approximately 25 times increase for GGAemx, 30 times increase for GGAdpx and even 70 times increase for Concorde. We should note also, that GGAemx and GGAdpx performed approximately the same, which shows that the recombination operator has no major influence on a final result.

In the second experiment we studied the influence of grafting (hybridization) on running time and quality of solution. In this experiment we used only grafted GA with edge map crossover (GGAemx) and only eleven cases from a TSPLIB (cf. Table 3.2). In the experiment we were increasing the number of generations, in which

Table 3.2: Eleven different levels of hybridization, which include canonical genetic algorithm (GAemx) and totally hybridized genetic algorithm (GGAemx). For each level of hybridization, except for GAemx and GGAemx, we applied local optimization in three different ways: at random generations (rnd), at the initial generations (begin) and at the ending ones (end).

Name	**GAemx**		**10**				**20**				**30**				**40**				**50**			
			rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a
	q	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*
eil76	8.93%	0.8	2.46%	2.13%	1.25%	1.2	1.80%	0.99%	0.96%	1.5	1.62%	0.92%	0.77%	1.8	1.58%	0.44%	0.22%	2.2	1.14%	0.37%	0.18%	2.6
pr76	5.39%	0.9	0.60%	0.41%	0.34%	1.3	0.32%	0.24%	0.19%	1.7	0.25%	0.17%	0.10%	2.1	0.20%	0.16%	0.09%	2.4	0.17%	0.11%	0.07%	2.7
gr96	6.46%	1.7	1.68%	0.78%	0.70%	2.3	1.37%	0.59%	0.55%	2.9	0.94%	0.59%	0.55%	3.6	0.78%	0.59%	0.55%	4.2	0.82%	0.55%	0.47%	4.9
rat99	6.14%	1.9	2.53%	1.82%	1.62%	2.6	2.67%	0.90%	0.71%	3.3	2.81%	0.62%	0.56%	4.1	2.13%	0.53%	0.39%	5.2	1.28%	0.43%	0.38%	6.3
kroA100	6.67%	0.6	1.09%	0.73%	0.38%	0.9	0.84%	0.38%	0.33%	1.2	0.19%	0.22%	0.12%	1.5	0.18%	0.08%	0.03%	1.8	0.16%	0.03%	0.02%	2.1
kroB100	7.02%	0.8	1.61%	1.15%	1.02%	1.3	1.26%	0.70%	0.53%	1.8	0.72%	0.70%	0.42%	2.3	0.71%	0.47%	0.35%	2.8	0.76%	0.40%	0.38%	3.3
kroC100	6.61%	0.7	2.20%	1.05%	0.99%	1.2	1.19%	0.90%	0.75%	1.6	0.97%	0.63%	0.52%	2.1	0.79%	0.44%	0.37%	2.5	0.74%	0.38%	0.36%	2.9
kroD100	7.67%	0.8	2.20%	1.87%	2.11%	1.3	2.39%	2.02%	1.47%	1.8	1.44%	1.17%	0.97%	2.3	1.26%	0.89%	0.67%	2.8	0.97%	0.54%	0.46%	3.3
lin105	8.54%	0.5	1.50%	1.11%	1.19%	0.9	0.89%	0.70%	0.50%	1.4	0.83%	0.40%	0.41%	1.8	0.71%	0.37%	0.29%	2.2	0.46%	0.23%	0.23%	2.6
ch150	8.69%	5.4	2.94%	2.52%	2.34%	6.2	2.17%	1.89%	1.83%	6.9	1.77%	1.58%	1.37%	7.8	1.63%	1.46%	1.36%	8.7	1.31%	1.19%	0.92%	9.6
***pr439*	10.45%	3.7	4.92%	4.35%	3.48%	11	4.59%	3.43%	2.96%	18	4.04%	3.16%	2.81%	25	3.34%	2.92%	2.56%	36.8	3.62%	3.13%	2.45%	45

	GGAemx		**90**				**80**				**70**				**60**			
			rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a	rnd	begin	end	f.a
	q	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*	*q*	*q*	*q*	*t*
eil76	0.04%	4.5	0.15%	0.04%	0.04%	4.1	0.18%	0.07%	0.04%	3.7	0.44%	0.15%	0.11%	3.3	1.07%	0.22%	0.11%	2.9
pr76	0.04%	4.1	0.07%	0.04%	0.04%	3.8	0.12%	0.10%	0.05%	3.5	0.11%	0.10%	0.06%	3.2	0.15%	0.11%	0.06%	2.9
gr96	0.12%	8.4	0.23%	0.16%	0.12%	7.7	0.27%	0.23%	0.20%	7	0.39%	0.35%	0.27%	6.3	0.74%	0.35%	0.31%	5.6
rat99	0.00%	11.9	0.07%	0.00%	0.00%	10.8	0.56%	0.05%	0.02%	9.7	0.61%	0.16%	0.05%	8.5	1.13%	0.30%	0.25%	7.4
kroA100	0.00%	3.6	0.00%	0.00%	0.00%	3.3	0.00%	0.00%	0.00%	3	0.01%	0.00%	0.00%	2.7	0.05%	0.00%	0.00%	2.4
kroB100	0.10%	5.8	0.22%	0.12%	0.09%	5.3	0.31%	0.37%	0.22%	4.9	0.52%	0.20%	0.24%	4.3	0.81%	0.30%	0.29%	3.7
kroC100	0.23%	5.3	0.37%	0.32%	0.23%	4.8	0.57%	0.56%	0.22%	4.3	0.52%	0.34%	0.29%	3.7	0.55%	0.32%	0.32%	3.3
kroD100	0.08%	5.6	0.32%	0.28%	0.08%	5.2	0.58%	0.31%	0.26%	4.7	0.61%	0.45%	0.40%	4.3	0.88%	0.53%	0.45%	3.8
lin105	0.10%	4.6	0.12%	0.09%	0.10%	4.2	0.11%	0.15%	0.10%	3.8	0.13%	0.12%	0.09%	3.4	0.21%	0.17%	0.12%	3.0
ch150	0.30%	15.2	0.81%	0.35%	0.30%	14.1	0.86%	0.42%	0.37%	13	0.96%	0.69%	0.54%	12	1.16%	0.97%	0.84%	10.7
***pr439*	1.30%	91.8	1.72%	1.66%	1.34%	78.9	2.28%	2.14%	1.97%	70	2.78%	2.18%	2.03%	62	3.26%	2.91%	2.31%	52.4

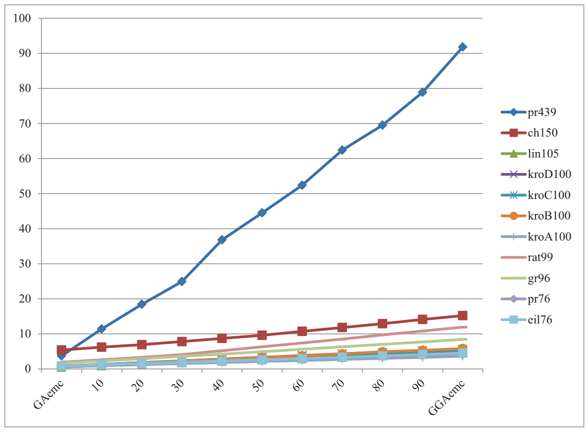

Figure 3.1: The running time as a function of level of hybridization.

we applied hybridization by 10 percents, i.e. level of hybridization: from 0% – column *GAemx* in Table 3.2 till 100% – column *GGAemx*. Moreover, we also varied the generations in which we applied the hybridization, i.e. place of hybridization: either in random generations (column *rnd*), in the beginning ones (*begin*) or in the ending ones (*end*). The column *f.a* gives the running time for all places of hybridization, while the numbers in columns q give the quality computed by using Equation 3.2.

We first observe, that application of grafting in the last generations gives the best results. This is reasonable, as in general in this phase of meta-heuristics we apply mostly intensification and not that much more diversification. On the other hand it is interesting that worse results were obtained when grafting was applied in random generations. Nonetheless, since in practice the algorithm does not know which are the last generations, we would need to simulate the behavior. There are two possibilities how to do it: either, when time limit is reached run the algorithm for some more runs and apply hybridization or apply hybridization more and more frequently as the number of generations increases. The later approach is also in line with other meta-heuristics like simulated annealing.

The running time obviously linearly increases as we increase the amount of hybridization (see Figure 3.1). Note, that even small hybridization of 10% drastically improves solution – e.g. for the *pr439* case to only about 4% off the optimal. On the other hand, further hybridization keeps improving result and it is up to the user to decide, how much hybridization should be employed.

Probably the decision on the amount of hybridization should be made considering the running time. As seen in Figure 3.1 the number of sites increases the steepness of the function. Therefore high hybridization at large cases would probably increase the running time substantially.

Additionally, we conducted statistical analysis of the evaluation results, where we analyzed the differences in level and place of hybridization for all examples together. The statistical analysis was performed on all the examples together, since the quality measure q, which is used to compute the evaluation results, was considered to be independent of the size of the problem. Figure 3.2 shows the box-plots of the results for all the cases subject to level of hybridization.

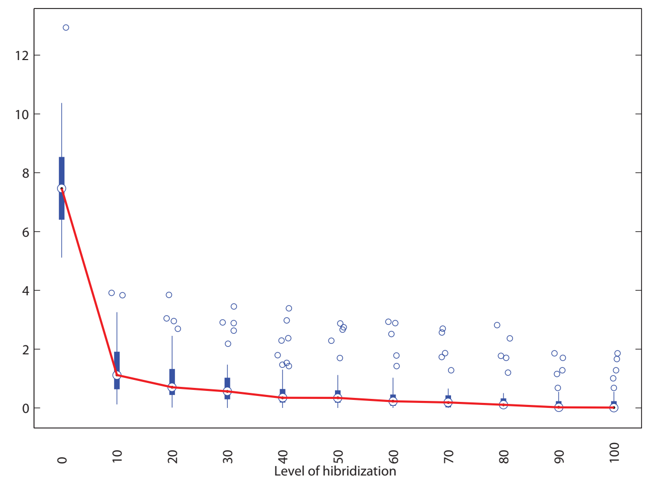

Figure 3.2: Quality of solutions: All cases

We firstly observe that the case with no hybridization is significantly worse compared to all other cases in the Figure 3.2. Next, we find that the quality of solution also improves as the level of the hybridization increases. From our experiments it seems to follow, that also lower amounts of hybridization give satisfactory results.

The statistical analysis of significance for place of hybridization was also performed for every tested level of hybridization. The analysis was performed by checking significance differences in evaluation scores, where the place of hybridization was set to random, or to begin or end. The statistical analysis was performed by applying one-way ANOVA test. Since the evaluation scores were not normally distributed, the nonparametric *Kruskal-Wallis Test* (KWT) was used instead of the standard one-way ANOVA. The differences were considered to be statistically significant in cases where the estimated *p-values* of statistical tests were less than or equal to 0.05.

The analysis with KWT showed that there exist statistically significant differences in place of hybridization. The analysis revealed significant differences in all levels of the hybridization. However, intensifying the level of hybridization further increased the p-value of the KWT, which means that the place of hybridization becomes of a less importance when the level of hybridization increases. In Figures 3.3 and 3.4 the results of KWT analysis are shown for place of hybridization at level 10% and 90%, respectively.

In Figure 3.3 we can observe that random place of hybridization significantly deviates from begin and end. This was confirmed by the KWT, which results to p-value of 0.0003 for 10% level of hybridization. Moreover, in Figure 3.4 we presents the result for 90% level of hybridization. In the illustrated case, a difference between places of hybridization is no more substantial. This was confirmed by KWT, which results to p-value approximately equal to 0.05. It was just at the upper margin of the significance level.

For the final analysis we show the evaluation results for the biggest tested case

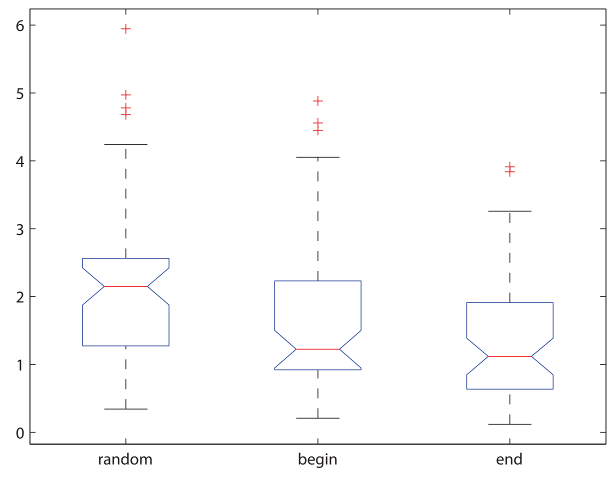

Figure 3.3: Level of hybridization 10%: All cases

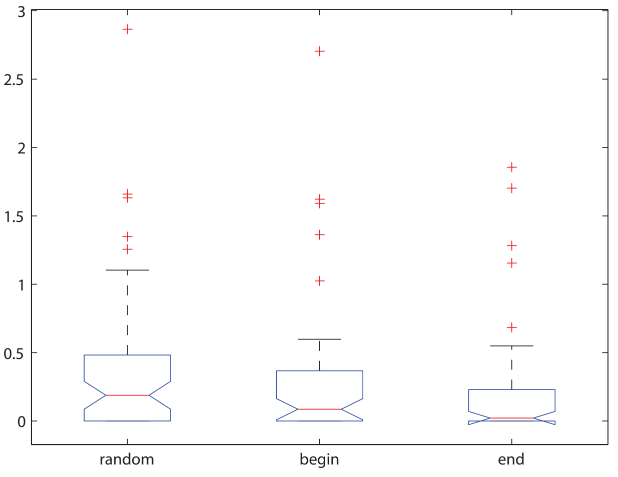

Figure 3.4: Level of hybridization 90%: All cases

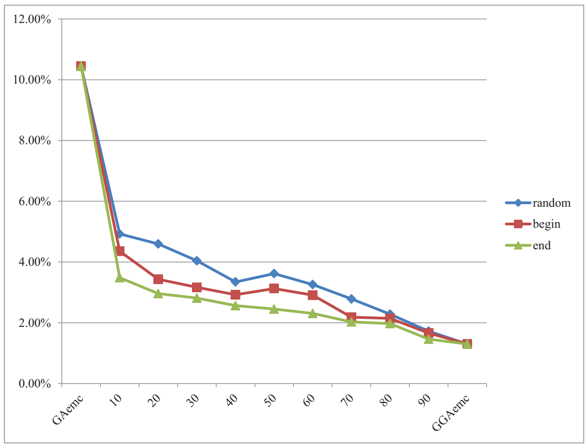

Figure 3.5: Quality results for case *pr439* as the level of hybridization increases

pr439 with 439 sites in Figure 3.5. The quality of solution improved from over 10% at no hybridization to approximately 3% at half hybridization and to 1.3% off the optimal at total hybridization. The result for half hybridization (50% level of hybridization) was significantly worse compared to the result of total hybridization. This was confirmed by Wilcoxon rank sum test, which amounts to p-value of 0.017.

3.5 Conclusions

The goal of this chapter was to investigate influence of grafting a 2-opt based local searcher into the canonical genetic algorithm, for solving the TSP. It is known that genetic algorithms are successful in solving many NP-hard problems. However, they are much more effective if some specific knowledge about particular problem is utilized. In our first experiment we compared two direct techniques with our grafted genetic algorithms. Solutions from Concorde and greedy algorithm were added for better comparison. Quantitative results on test cases from *TSPLIB* show that grafted algorithms have advantages. Even when both components have serious drawbacks, their grafted combinations exhibits a very good behavior. Results on examples from *TSPLIB* show that this method combines good qualities from both methods and outperforms each individual method.

Our experiments further show that the best results are obtained when hybridization occurs in the last generations of the GA. This seems to be in line with classical meta-heuristic algorithms such as simulated annealing, which stop their diversification in the last iterations. We showed that even a small hybridization substantially improves the quality of the result. Moreover, the hybridization in fact does not deteriorate the running time too much.

Chapter 4

Traveling Visitor Problem

4.1 Introduction

In the Traveling Salesman Problem (TSP) a set $\{c_1, c_2, ...c_n\}$ of cities is considered and for each pair (c_i, c_j) where $i \neq j$, a distance $d(c_i, c_j)$ is given. The goal is to find a permutation π of the cities that minimizes the quantity

$$\sum_{i=1}^{n-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(n)}, c_{\pi(1)}). \tag{4.1}$$

This quantity is referred to as the tour length since it is the length of the tour a salesman would have to travel when visiting the cities in the order specified by the permutation π, returning at the end to the initial city. The TSP is known to be *NP-hard* [72]. The case with symmetric distances has been well studied and there are many algorithms which perform well even on large cases [3, 5]. In the literature the TSP is usually represented and considered as a graph theoretical problem, see [60,71]. For more details about TSP see Section 2.1.

An instance of the symmetric TSP can be represented as a complete graph $G = (V, E)$ with the set of vertices V (cities) and set of edges between cities with corresponding edge weights $d(c_i, c_j)$. The symmetric TSP translates to the problem of finding a Hamiltonian Tour of minimal length in the undirected graph G.

Applications of the TSP and its variations go way beyond the route planning problem of a traveling salesman and span over several areas of knowledge including mathematics, computer science, operations research, genetics, engineering, and electronics. In addition, there are many different variations of TSP which are described and explored in the literature and also variations derived from everyday life. Some of them are: *machine scheduling problems* [8, 60], the *time dependent TSP* [57], the *delivery man problem* which is also known as the *minimum latency problem* and the *traveling repairman problem*, for details on these problems, we refer to Section 2.1.3.

The Traveling Tourist Problem [90] is a problem in which a tourist wishes to see all monuments (nodes) in a city, and so must visit each monument or a neighbor thereof. The Traveling Tourist Problem shares a similar name with our problem however it is a very different problem.

The symmetric TSP can be solved using several different methods including the Grafted Genetic Algorithms (GGA) as was shown in [34–37] and in Section 3. The currently most efficient implementation of the branch-and-cut method introduced by Padberg and Rinaldi [110] for solving the symmetric TSP is named *Concorde* [4]. *Concorde's* TSP solver has been used to obtain the optimal solutions to the full set of 110 TSPLIB instances, the largest having 85,900 cities. For more details see Section 2.2.5.

Finally, in a graph G besides finding the shortest closed walk we can also look for the shortest path between any pair of vertices. This problem is in the literature known as *all-pairs shortest path problem* (APSP) [27]. The Floyd-Warshall algorithm [27] is an efficient algorithm to find all-pairs shortest paths on a graph G. The all-pairs shortest path problem and the Floyd-Warshall algorithm are in more details described in Section 2.1.5.

4.2 Traveling Visitor Problem

Suppose that a visitor arrives in a hotel in a town and wants to visit all interesting sites in the city exactly once and to come back to the hotel at the end of her journey. Visitor in general moves from one place to another using the streets, walking trails and pedestrian zones. The goal is to minimize the visitors traveling distance.

The Traveling Visitor Problem is a version of the TSP with additional constraint that she must go around these obstacles (buildings), see Figure 1.1. Formally, the Traveling Visitor Problem (TVP) is defined as:

Definition 4.2.1 *Given a connected, weighted graph $G = (V, E, W)$, with a set of vertices $V = S \cup X$ and $S \cap X = \emptyset$, where S is the set sites of interest (vertices u and v in Figure 1.1), and X is the set of nodes representing crossroads in the city (vertices a and b in Figure 1.1). Further, E a set of edges, and W is the distance matrix, i.e. a cost of traveling. The goal is to find the shortest closed walk of simple visits of all vertices from S, although we may travel through vertices from X arbitrary number of times.*

The concepts we summarized above can be modified easily to take the directions of the edges into account. The asymmetric TVP is then similar to the symmetric TVP above, i.e. it is the problem of finding a closed walk of minimal length in a weighted graph. So far, this problem, by the knowledge of the authors, has no references in publications at the date of writing it.

4.3 Algorithms for Solving TVP

The simplest approach of solving TVP is to visit all places as are ordered in the city's tourists maps and then come back to the starting site. The result of this method depends directly on the order in which the interesting sites are listed on the map and most probably does not produce the shortest closed walk through all sites of interest.

The first proposed method for solving the TVP is the Naïve algorithm, shown in Algorithm 10.

Algorithm 10 Naïve Algorithm

```
procedure NAÏVE(S,X,W)
    /* $S$= set of sites of interest; $X$= set of crossroads; $W$= distance matrix */
    $\pi \leftarrow TSP(S)$ /* TSP= Traveling Salesman Problem */
    $Z \leftarrow APSP(S, X, W)$ /* APSP= All-Pairs Shortest Path Problem */
    cost $\leftarrow 0$
    for all $(i, j) \in \pi$ : do
        cost $\leftarrow$cost $+Z_{ij}$
    end for
end procedure
```

The parameter S is the set of sites of interest, X is the set of crossroads and W is the edge distance matrix of the graph G, $(S \cup X \times S \cup X)$. The algorithm first assumes that sites from S are on an Euclidean plane with no obstacles. It solves TSP for such a set of points (line 3 in Algorithm 10). It then searches for the closest path between consecutive points of the obtained travel π avoiding obstacles. The later is done by computing shortest distance between points of S but possibly passing through X (line 4 in Algorithm 10) obtaining matrix Z. Finally, we calculate the final cost combining the obtained tour π with the shortest paths from Z in lines 6 through 8.

However, our problem of shortest paths is somewhat more restrictive as V consists of disjoint subsets X and S and we are only interested in the shortest paths between vertices of S while a path itself can visit an arbitrary number of vertices in X. This was the reason to adjust the Floyd-Warshall algorithm for solving the APSP.

4.3.1 Adapted Floyd-Warshall Algorithm

In Section 2.1.5 we met a classical all-pairs shortest path problem in graph $G = (V, E)$, which can be solved using Floyd-Warshall algorithm (cf. Algorithm 1, where W is the distance matrix of the graph G) in time $|V|^3$). The restriction presented in previous section permits us to modify slightly the Floyd-Warshall algorithm and obtain the following Lemma.

Lemma 4.3.1 *Let $G = (X \cup S, E)$ be a graph where $X \cap S = \emptyset$, $x = |X|$ and $s = |S|$. Let a S-path be a sequence of edges from E that starts and ends in S but can also visit vertices in X. Then there exists an algorithm that computes all-pairs shortest S-path between any pair of $u, v \in S$ in time $s^3 + x^3 + s^2x + x^2s$. Considering also the leading term, this can be up to twice as fast as Floyd-Warshal algorithm.*

PROOF. We split the graph G into several subgraph components consisting of vertices either from S or X, and edges between these components. Figure 4.1 shows an example of splitting S and X into components S' and S'', and components X' and X'' respectively ($S = S' \cup S''$, $X = X' \cup X''$, where $S' \cap S'' = \emptyset$, $X' \cap X'' = \emptyset$). To illustrate the proof we use the example from the figure. Although in general S and X can break in several components, it will be easy to see that the number of

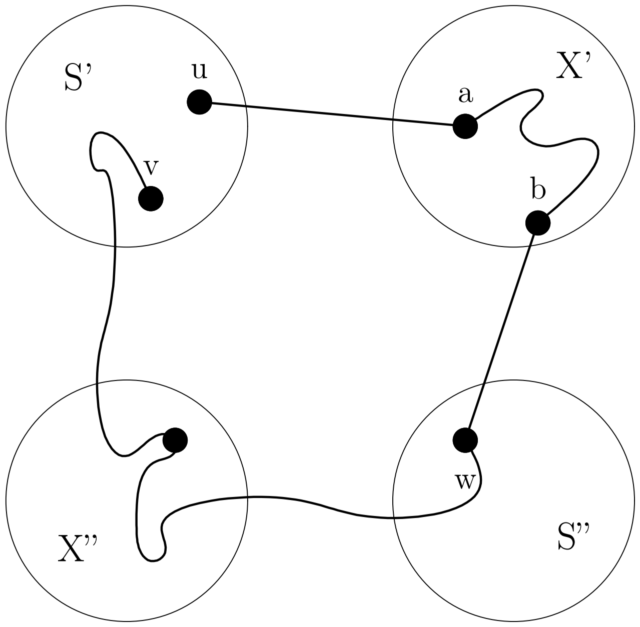

Figure 4.1: Each node represents connected component consisting of vertices only from either X or S, while edges represent (arbitrarily many) connections between components

components does not influence neither correctness nor the time complexity of our algorithm.

Consider the path π_{uv} between nodes $u, v \in S$. It starts and ends in S, but it can pass through X (cf. Figure 4.1). Moreover, it can oscillate between X and S. Assume that π_{uv} for the first time re-enters S at vertex $w \in S$. Due to the minimization principle, paths π_{uw} and π_{wv} are also the shortest paths between u and w, and w and v respectively. Therefore, if we compute these paths first, the relaxation principle in Floyd-Warshall algorithm will pick up the correct path for π_{uv}.

Furthermore, let $\pi_{uw} = u \to \pi_{ab} \to w$, that is path π_{uw} enters X at a and exits at b. As before π_{ab} is also the shortest path between $a, b \in X$, which is completely in X. Similarly paths $u \to \pi_{ab}$ and $\pi_{ab} \to w$ are also the shortest paths between the respective vertices. Therefore, if we have computed the last two paths, the relaxation procedure will pick the correct path when computing the shortest path π_{uw}

This brings us to the algorithm in Algorithm 11. We first compute all-pairs shortest paths (cf. π_{ab}) between all vertices in X, which takes x^3 time. Next (lines 4-10), we compute the shortest paths between all nodes $a \in X$ and $w \in S$ (cf. $\pi_{ab} \to w$), which takes another $x^2 s$ time. In lines 11-17 we compute the shortest paths between all nodes $u \in S$ and $b \in X$ (cf. $u \to \pi_{ab}$), which takes another xs^2 time. The final call to Floyd-Warshall algorithm on S in time s^3 computes the final result as all possible sub-paths are already computed. The total time of the our algorithm is $x^3 + x^2 s + xs^2 + s^3$ which has extreme $4s^3$ at $x = s$. On the other hand the usual Floyd-Warshall algorithm takes in this case $(x+s)^3 = 8s^3$ ■

Algorithm 11 Adapted Floyd-Warshall Algorithm

```
1: procedure ADAPTED(S,X,W)
2:     /* $S$= set of sites of interest; $X$= set of crossroads; $W$= distance matrix */
3:     FLOYD-WARSHALL($X, W$)
4:     for all $k \in X$ do
5:         for all $i \in X$ do
6:             for all $j \in S$ do
7:                 $W_{ij} := min(W_{ij}, W_{ik} + W_{kj})$
8:             end for
9:         end for
10:    end for
11:    for all $k \in X$ do
12:        for all $i \in S$ do
13:            for all $j \in S$ do
14:                $W_{ij} := min(W_{ij}, W_{ik} + W_{kj})$
15:            end for
16:        end for
17:    end for
18:    FLOYD-WARSHALL($S, W$)
19: end procedure
```

.

4.3.2 Koper Algorithm

In the Naïve algorithm (cf. Algorithm 10) we first solve a TSP on Euclidean plane using the shortest direct connections between the sites obtaining the first approximation and later solve the restricted version of APSP problem. However, the tour obtained in the first step might be incorrect due to obstacles and this was a reason to transform the Naïve algorithm into new one.

The second proposed method for solving the TVP is Koper algorithm, shown in Algorithm 12. In Koper algorithm we first compute all-pairs shortest paths in graph G using the Adapted Floyd-Warshall algorithm, shown in Algorithm 11 (line 3 in the Algorithm 12) obtaining a distance matrix Z between points of S. In the next step we solve the TSP, using the distance matrix Z obtaining the tour π, which is a solution for TVP (line 5 in the Algorithm 12).

Algorithm 12 Koper Algorithm

```
1: procedure KOPER(S,X,W)
2:     /* $S$= set of sites of interest; $X$= set of crossroads; $W$= distance matrix */
3:     $Z \leftarrow ADAPTED(S, X, W)$
4:     /* ADAPTED= Adapted Floyd-Warshall Algorithm */
5:     $\pi \leftarrow TSP(Z)$ /* TSP= Traveling Salesman Problem */
6: end procedure
```

4.4 Experiment

For testing we used the real instances of the TVP, which were made from official tourist maps of cities Koper, Belgrade and Venice. In the Belgrade example two different cases were considered with a different number of vertices in the graph. From the library, TSPLIB, we selected two instances of the symmetric TSP and modified them into TVP instances.

These two instances were modified in such a way that a new connected graph G' was constructed. Furthermore, we split V into a set of vertices S and X, such that $|S| = |X| = |V|/2$. The degree of vertices was chosen to be 5 inspired by the real instances. The 5 edges per vertex were chosen randomly, according to a uniform probability distribution.

Altogether 5 instances were tried out, with different sizes ranging from 120 to 1002 vertices per instance. We compared two methods for solving the TVP. The Naïve algorithm (cf. Algorithm 10) and the Koper algorithm (cf. Algorithm 12). For solving the TSP, as a step in both algorithms, we used the Concorde Algorithm, presented in Section 2.2.5.

4.5 Evaluation

The results of the experiment are summarized in Table 4.1. The names of instances are in the first column. The second and the third columns contain the size of the problem, i.e. the number of vertices in set V and the number of vertices in set S respectively. The next two columns contain the method and tour length. Last column shows the quality of the result computed by formula

$$q_N = \frac{l_N - l_K}{l_K}, \tag{4.2}$$

where l_K is a length of the tour obtained by Koper algorithm and l_N is a length of the tour obtained by Naïve algorithm. The relative difference is expressed always in percents. For example, 17.22% difference means that Naïve algorithm performed 17.22% worse than Koper algorithm. The first tested method, the Naïve algorithm, performed poorly in comparison to the Koper algorithm. The quality differs from 6.52% in the case of Belgrade163 to 354.46% in the case of pr1002 instance. The difference in the quality is bigger in instances that are more complex.

Although these algorithms are similar in terms of components (both rely on solving an APSP and TSP), the difference on the quality of the solutions indicates that there is a gain by using the Koper algorithm. Note also that the running time for both algorithms is the same.

4.6 Conclusions

The goal of this chapter is to describe a new problem from graph theory, named the Traveling Visitor Problem. Although the new problem is similar to the Traveling Salesman Problem, when we try to solve it with the Naïve algorithm we get solutions

Name	(V)	(S)	Methods	Tour Cost	Difference
Koper	120	55	Naïve	4738	17.22%
			Koper	4042	
Belgrade	163	53	Naïve	100389	6.52%
			Koper	94246	
	250	90	Naïve	122119	8.77%
			Koper	112275	
Venice	210	72	Naïve	26648	24.24%
			Koper	21448	
lin318	318	159	Naïve	921499	249.08%
			Koper	263983	
pr1002	1002	501	Naïve	11818732	354.46%
			Koper	2600585	

Table 4.1: Two techniques for solving the Traveling Visitor Problem

far from optimal. The minimum cost solutions for the TVP instances tested in the chapter are provided by our Koper algorithm. The tested benchmarks are obtained from three real instances coming from tourist maps of cities of Koper, Belgrade and Venice and two modified instances from TSPLIB. In all tested cases the Koper algorithm outperforms the Naïve algorithm for solving the TVP - quality of solutions differs from 6.52% to 354.46%.

Chapter 5

Conclusion

The first goal of the thesis was to investigate influence of grafting a 2-opt based local searcher into the standard genetic algorithm, for solving the Traveling Salesman Problem. It is known that genetic algorithms are successful when implemented for many NP-hard problems. However, they are much more effective if some specific knowledge about particular problem is utilized. In our experiment in Chapter 3 we compared two canonical techniques, with our grafted genetic algorithms. For the reference, solutions from Concorde, state of the art TSP solver, and greedy algorithm were added for better comparison. Quantitative results on test cases from *TSPLIB* show that grafted algorithms have advantages. Even when both components have serious drawbacks, the grafted versions show very good behavior. Results on examples from *TSPLIB* show that grafting (hybridization) combines good qualities from both methods applied and outperforms each individual method. Our experiments further show that the best results are obtained when hybridization occurs in the last generations of the GA. This seems to be in line with classical meta-heuristic algorithms like simulated annealing, which stops its diversification in the last iterations. On one hand the less frequent application of hybridization decreased the average running time of the algorithm from 14.62 sec to 2.78 sec at 100% and 10% hybridization respectively, while on the other hand the quality of solution on average deteriorated only from 0.21% till 1.40% worse than the optimal solution. We showed, that even a small hybridization substantially improves the quality of the result. Moreover, the hybridization in fact does not deteriorate the running time too much.

The second goal of the thesis was to study a new problem from graph theory, named the Traveling Visitor Problem. Although the new problem is similar to the Traveling Salesman Problem, when we try to solve it in the same way using the Naïve algorithm we get solutions far from optimal. The minimum cost solutions to the Traveling Visitor Problem instances in the paper are provided by the Koper Algorithm. The tested benchmarks used come from three real instances made using tourist maps of cities of Venice, Belgrade and Koper and two modified instances from TSPLIB. In all tested cases the Koper Algorithm significantly outperforms the Naïve Algorithm for solving the Traveling Visitor Problem - quality of solutions differs from 6.52% to 354.46%.

5.1 Hypotheses and Contribution to the Science

In chapter 1 two hypotheses was introduced:

- Hypothesis 1: The method for solving of TSP that is made of two independent methods, genetic algorithm and 2-opt heuristic, outperforms each of the combined methods in terms of the quality of solution.
- Hypothesis 2: The quality of solution of a proposed method to the problem of a traveling visitor problem, outperforms algorithms for solving general TSP problem when they are used for solving traveling visitor problem.

Using a specific methodology, in detail described in Chapter 3, we prove the correctness of the hypothesis 1. Furthermore, in Chapter 4 we prove the correctness of the hypothesis 2.

Contributions to the science consist of the following results:

- construction of the grafted genetic algorithm for solving the traveling salesman problem,
- insights of different levels and places of hybridization for grafted genetic algorithm,
- construction of the method for solving the traveling visitor problem,
- construction of improved Floyd-Warshall algorithm for all-pairs shortest path problem,
- solutions of the traveling visitor problem for cities of Koper, Belgrade and Venice.

The results of this PhD Thesis are published in the following articles:

- M. Djordjevic, Influence of Grafting a Hybrid Searcher Into the Evolutionary Algorithm, *Proceedings of the 17th International Electrotechnical and Computer Science Conference, Portoroz, Slovenia* (2008), 115–118.
- M. Djordjevic, M. Tuba, and B. Djordjevic, Impact of Grafting a 2-opt Algorithm Based Local Searcher Into the Genetic Algorithm, *Proceedings of the 9th WSEAS international conference on Applied informatics and communications, AIC 2009, Moscow, Russia* (2009), 485–490.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Book of Abstract of International Conference on Operations Research, OR 2011, Zurich, Switzerland* (2011), 130.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011, Dubrovnik, Croatia* (2011), 515–520.

- M. Djordjevic, M. Grgurovic and A. Brodnik, The Traveling Visitor Problem and the Koper Algorithm for Solving It, *Book of Abstracts of 25th Conference of European Chapter on Combinatorial Optimization, ECCO 2012, Antalya, Turkey* (2012), 10.
- M. Djordjevic, M. Grgurovic and A. Brodnik, The Traveling Visitor Problem and Algorithms for Solving It, *Book of Abstracts of 3rd Student Conference on Operational Research, SCOR 2012, Nottingham, UK* (2012), 26.
- M. Djordjevic, J. Zibert, M. Grgurovic, and A. Brodnik Methods for Solving the Traveling Visitor Problem, *Proceedings of the 1st International Internet and Business Conference, IBC 2012, Rovinj, Croatia* (2012), 174–179.
- M. Djordjevic, M. Grgurovic, and A. Brodnik, Performance Analysis of Partial Use of Local Optimization Operator on Genetic Algorithm for Traveling Salesman Problem, *Business Systems Research*, Print ISSN 1847-8344; Online ISSN 1847-9375, in press.

Povzetek v slovenskem jeziku

5.2 Uvod

Leta 2007 sem se vpisal podiplomski doktorski študij računalništva v Kopru. Iz Beograda sem prišel z veliko željo spoznati mesto, v katerem sem nameraval živeti naslednja štiri leta. Mestno jedro mi je bilo takoj všeč, predvsem zaradi številnih znamenitosti, natančno 55 jih je, ki se nahajajo tudi na turističnem zemljevidu Kopra. Doktorski študij je naporen in ker nisem imel veliko prostega časa, sem se začel spraševati, kako bi bilo, če bi lahko optimiziral svoj obhod mestnih znamenitosti na tak način, da bi porabil najmanj možnih korakov in si tako prihranil nekaj časa. Problem sem poimenoval *problem potujočega obiskovalca* (angl. *Traveling Visitor Problem, (TVP)*).

Problem potujočega obiskovalca TVP smo v doktorski disertaciji reševali preko problema trgovskega potnika (angl. *Traveling Salesman Problem, (TSP)*). Problem TSP je v zgodnjih 30. letih 20. stoletja predstavil avstrijski matematik Karl Menger [94] kot problem glasnika, ki želi obiskati vsako mesto s seznama, na katerem je n mest, natanko enkrat in se nato vrniti v svoje mesto, pri čemer so cene potovanj iz mesta i v mesto j znane vnaprej. Vprašanje je torej, kateri od obhodov je najcenejši? Problem TSP je formalno definiran na polnem grafu $G = (V, E)$, kjer je $V = \{v_1, v_2, ..., v_n\}$ množica vozlišč, E množica povezav s cenilno funkcijo $C(i, j)$, ki povezavi $(i, j) \in E$ priredi določeno ceno.

TSP lahko obravnavamo tudi kot problem iskanja permutacij. Naj bo P_n množica vseh permutacij iz množice $\{1, 2, ..., n\}$. Potem je *problem trgovskega potnika* poiskati $\pi = (\pi(1), \pi(2), ..., \pi(n))$ v P_n, za katero velja, da je $c_{\pi(n)\pi(1)} + \sum_{i=1}^{n-1} c_{\pi(i)\pi(i+1)}$, minimalen.

TSP je eden izmed najpomembnejših predstavnikov večje množice problemov, imenovane kombinatorični optimizacijski problemi [63]. Ker sodi TSP v razred NP-težkih (angl. *NP-Hard*) problemov [72], ne poznamo učinkovitega algoritma za reševanje TSP. Natančneje, takšen algoritem obstaja le pod pogojem, da sta razreda P in NP enaka. S praktičnega vidika to pomeni, da ne poznamo natančnega algoritma za kateri koli TSP-primer z n vozlišči, ki se obnaša znatno bolje kot algoritem, ki izračuna vseh $(n-1)!$ možnih obhodov ter vrne obhod z najmanjšo ceno.

V praksi lahko za reševanje tega problema uporabimo tudi drugačen pristop. Določeni primer TSP z n vozlišči ima lahko kateri koli obhod, ki poteka skozi vsa vozlišča n, in predstavlja možno rešitev, t.j. zgornjo mejo (angl. *upper bound*) za naj nižjo možno ceno. Algoritem, ki v polinomskem času (angl. *polynominal time*) konstruira možne rešitve za to zgornjo mejo, se imenuje hevristika [12, 115]. Načeloma, ti algoritmi tvorijo rešitve, vendar brez zagotovila o kakovosti rešitve glede na razliko med njihovo ceno in optimalno ceno.

Poznamo dve vrsti TSP: simetrični TSP in asimetrični TSP. V simetrični obliki, znani pod imenom STSP [69, 70, 88, 92], je razdalja med vozliščema i in j enaka razdalji med vozliščema j in i. V primeru asimetričnega TSP (ATSP) [14,16,18,23], takšna simetrija ne obstaja. Poleg tega obstaja še vrsta različnih variacij TSP, ki so opisane in raziskane v literaturi ter predstavljene v doktorski disertaciji v drugem poglavju. Te so:

Gručni TSP (angl. *Clustered TSP*) [61], ozko-grlni TSP (angl. *Bottleneck TSP*)-[60], posplošeni TSP (angl. *Generalized TSP*) [59, 75], problem glasnika (angl. *Messenger Problem*) [94], ki je znan tudi kot problem izgubljenega prodajalca (angl. *Wondering Salesman Problem*) [74], problem zamenjave (angl. *The swapping problem*) [2], problem minimalne latentnosti (angl. *Minimum Latency Problem*) [11], poznan še kot problem dostavljavca (angl. *Delivery Man Problem*) [60] ali problem potujočega mojstra (angl. *Traveling Repairman Problem*) [52], problem seizmičnih plovil (angl. *Seismic Vessel Problem*) [58], ki je posplošitev problema skladiščnega dvigala (angl. *Stacker Crane Problem*) [24], problem potujočega turnirja (angl. *Traveling Tournament Problem*) [41], problem lokacije objekta (angl. *Facility Location Problem*) [40]; in končno problem potujočega obiskovalca, ki je podrobno opisan, raziskan in razrešen v doktorski disertaciji, v poglavju 4. Ta problem predhodno ni bil naveden v kakršni koli literaturi.

Prvi koraki v reševanju TSP so bili klasični poskusi z metodami, sestavljenimi iz natančnih in hevrističnih algoritmov. Natančne metode, kot so presek ravnine (angl. *cutting planes*) [31], razveji in omeji (angl. *branch and bound*) [26, 31], lahko optimalno rešijo relativno majhne probleme (v odvisnosti od velikosti n), medtem ko nam dajo metode, kot so različne variante algoritma Lin-Kernighan [6, 45, 66, 75] in tehnike Concorde [3–5] relativno dobre rezultate tudi za večje probleme. Za reševanje TSP se prav tako uporabljajo algoritmi, zasnovani na *požrešnih* metodah, kot sta najbližji sosed (angl. *nearest neigbour*) [60] in vpeto drevo (angl. *spanning tree*) [65].

Računska kompleksnost natančnih metod za reševanje TSP je eksponentna, zato je bilo potrebno razviti boljše rešitve. Te vključujejo različne principe optimizacijskih tehnik, naravno orientirane optimizacijske algoritme, populacijsko orientirane optimizacijske algoritme ter druge. Nekatere od teh metod so predstavljene v doktorski disertaciji v poglavju 2: evolucijsko računanje (angl. *Evolutionary Computation*) [95, 100, 121, 130], genetski algoritmi (angl. *Genetic Algorithms*) [42, 50, 51, 97, 103, 111, 120, 124, 128, 129, 133, 134], memetični algoritmi (angl. *Memetic Algorithms*) [59, 83–85, 98, 107], sistemi mravelj (angl. *Ant Systems*) [38], simulirano ohlajanje (angl. *Simulated Annealing*) [80], in nazadnje *cepljeni genetski algoritmi* (angl. *Grafted Genetic Algorithms*) [34–37], ki predstavljajo vrsto hibridnih genetskih algoritmov. Ti so podrobneje raziskani ter predstavljeni v doktorski disertaciji v poglavju 3.

5.3 Vsebina disertacije

Doktorska disertacija po Pravilniku o pripravi in zagovoru doktorske disertacije na Univerzi na Primorskem, vsebuje več poglavij:

- Povzetek
- Kazalo vsebine
- Poglavje 1 - Uvod
- Poglavje 2 - Ozadje
  - 2.1 Problem trgovskega potnika
  - 2.2 Optimizacijski algoritmi
- Poglavje 3 - Cepljeni genetski algoritmi
- Poglavje 4 - Problem potujočega obiskovalca
- Zaključek
- Literatura

V poglavju *Ozadje* smo predstavili osnovne pojme trgovskega potnika in optimizacijskih algoritmov, ki smo jih uporabili za reševanje problemov v doktorski disertaciji. Preostala poglavja so namenjeni predstavitvi doseženih ciljev disertacije. Poglavja so razdeljena na več podpoglavij.

V doktorski disertaciji sta obdelani dve temi s področja teoretičnega računalništva. Optimizacijsko-hevristična metoda, imenovana cepljeni genetski algoritmi in kombinatorično-optimizacijski problem, imenovan tudi problem potujočega obiskovalca. Cilja doktorske disertacije sta dva:

Razvoj cepljenih genetskih algoritmov: cilj je pokazati kakovost dobljenih rešitev in hitrost izvajanja cepljenega genetskega algoritma v primeru reševanja problemov simetričnih TSP-jev. Algoritmi so bili ovrednoteni na zbirki splošno priznanih ocenjevalnih primerov TSP-jev.

Reševanje problema potujočega obiskovalca: cilj je opisati in definirati problem potujočega obiskovalca, ustvariti realne primere in rešiti primere problemov z uporabo nove metode ter izvesti primerjavo z drugimi znanimi metodami.

Raziskovalni cilj disertacije je dokazati spodaj navedeni hipotezi 1 in 2.

Hipoteza 1: metoda za reševanje TSP, sestavljena iz dveh neodvisnih metod, genetskega algoritma in 2-opt hevristike, združuje kakovosti obeh metod na takšen način, da ju znatno prekaša glede na kakovost rešitve.

Hipoteza 2: v okviru doktorske disertacije razvita metoda za reševanje problema potujočega obiskovalca prekaša splošne algoritme za reševanje problema trgovskega potnika, ki jih uporabimo za reševanje problema potujočega obiskovalca.

5.4 Raziskava

5.4.1 Cepljeni genetski algoritmi

Botanično cepljenje je postopek, pri katerem je tkivo prve rastline pritrjeno na tkivo druge rastline. Cepljenje lahko skrajša čas do cvetenja in skrajša čas vzgoje cepljene rastline. Idejo cepljenja v botaniki smo uporabili tudi za izboljšavo genetskih algoritmov. Ena izmed možnih razširitev konvencionalnega genetskega algoritma z uporabo metode cepljenja (hibridizacije) je lokalno iskanje oziroma optimizacija (angl. *Local Searcher*). Lokalno iskanje je metoda, ki omogoča optimizacijo genoma izven evolucijskega procesa. Tako cepljen genetski algoritem je predstavljen v algoritmu 9. V algoritmu se po izvedeni rekombinaciji (vrstica 8 v algoritmu 9 uporablja lokalno iskanje za optimizacijo genoma potomcev (vrstica 9 v algoritmu 9). Zaradi uporabe omenjene zunanje optimizacijske metode govorimo o cepljenem genetskem algoritmu [34–37]. Omenjena oblika optimizacije se izvaja lokalno ter spreminja genom s pomočjo hevrističnega spreminjanja rešitve. 2-opt lokalna optimizacija, poglavje 2.2.3, je metoda za hevristično reševanje TSP, ki je bila vcepljena v standardni genetski algoritem (vrstica 9 v algoritmu 9). Ta metoda izvaja 2-opt hevristiko, ki izmenjuje povezave grafa z namenom zmanjšati dolžino obhoda. Postopek izmenjave se sestoji iz odstranjevanja dveh povezav iz trenutnega obhoda in ponovnega povezovanja na najboljši možen način, kot je to prikazano na sliki 2.1.

V opravljenem poskusu smo testirali vpliv cepljenja algoritma lokalnega iskanja z genetskim algoritmom za reševanje problema trgovskega potnika. Za testiranje naše strategije in njeno primerjavo z ostalimi rešitvami, smo uporabili primere simetričnega problema trgovskega potnika, ki so na voljo na spletu, v knjižnici TSPLIB [116]. Uporabili smo 20 primerov z različno kompleksnostjo in obsegom od 14 do 150 mest (tabela 3.1). Našo metodo (cepljeni genetski algoritem), katere predstavnika sta algoritma GGAemx in GGAdpx, smo primerjali s štirimi drugimi metodami. Za spodnjo mejo kakovosti rešitve smo uporabili požrešno hevristiko (angl. *Greedy Heuristic* 2.2.2), za zgornjo mejo globalni minimum, pridobljen s programom Concorde 2.2.5. Nato smo primerjali našo cepljeno metodo z 2-opt in genetskim algoritmom.

Rezultati eksperimenta so predstavljeni v tabeli 3.1. Šesti stolpec v tabeli prikazuje rešitve našega cepljenega algoritma, kjer je bil operator rekombinacije sprogramiran s pomočjo križanja povezav (angl. *edge map crossover*, poglavje 2.2.4), GGAemx.

V sedemnajstih primerih od dvajsetih obravnavanih je bila najdena optimalna rešitev, preostali trije primeri odstopajo od optimalne rešitve za 0,01, 0,10 in 0,22 odstotka. Rešitve so bile najdene hitro v času od 0,6 do 15,2 sekund, z relativno majhnim številom iteracij. V sedmem stolpcu v tabeli 3.1 so prikazani rezultati našega cepljenega genetskega algoritma, kjer se kot operator za rekombinacijo uporablja križanje ki ohranja razdaljo (angl. *distance preserving crossover*, poglavje 2.2.4), GGAdpx. V enajstih od dvajsetih obravnavanih primerov je bila najdena optimalna rešitev, v preostalih devetih primerih so odstopanja od optimalne rešitve od 0,13 do 0,32 odstotka. V primerjavi z GGAemx sta čas izvajanja in število generacij GGAdpx nekoliko manjša, posebej v kompleksnejših primerih.

Kvantitativni rezultati na testnih primerih iz TSPLIB kažejo na učinkovitost cepljenih algoritmov GGAemx in GGAdpx. Iskaže se, da se kljub številnim pomanjkljivostim njunih komponent, kombinacije cepljenja obnesejo kot učinkovite. Rezultati primerov iz TSPLIB kažejo, da omenjena metoda cepljenja združuje dobre lastnosti iz obeh uporabljenih metod in prekaša vsako izmed njiju.

5.4.2 Problem potujočega obiskovalca

Problem potujočega obiskovalca lahko predstavimo na sledeč način. Obiskovalci so prispeli v hotel v neko nepoznano mesto in želijo obiskati vse mestne znamenitosti natanko enkrat in se po ogledu vrniti v hotel. Na ogled so se odpravili peš po ulicah, sprehajalnih področjih in poteh za pešce. Cilj je po čim krajši poti obiskati vse znamenitosti natanko enkrat.

Problem potujočega obiskovalca je izpeljan iz problema trgovskega potnika, pri čemer velja pravilo, da obiskovalec izbira samo med potmi, ki jih je možno prehoditi. To pomeni, da poti, kot so definirane v evklidskem TSP [55, 107], kjer je pot definirana kot najkrajša pot med dvema točkama, v tem primeru niso vedno mogoče. Obiskovalci uporabljajo sprehajalne poti in območja za pešce različnih dolžin. Te omejitve določajo težo povezav, ki povezujejo vozlišča v grafu.

Definicija TVP je tako naslednja: imamo graf $G = (V, E, W)$, kjer je množica vozlišč $V = S \cup X$ in $S \cap X = \emptyset$, pri čemer so S mestne znamenitosti in X križišča, E množica povezav ter W matrika razdalj oz. cena potovanja. Cilj je poiskati najkrajši sklenjen obhod (angl. *closed walk*, poglavje 2.1.1) skozi vsa vozlišča S v grafu G, pri čemer se lahko sprehodimo skozi X.

Prva predlagana metoda za reševanje problema potujočega obiskovalca je naivni algoritem (angl. *Naïve Algorithm*), prikazan v algoritmu 10. V algoritmu imamo naslednje parametre: S pripada zanimivim lokacijam v mestu, X križiščem, W pa predstavlja matriko povezav grafa G, $(S \cup X) \times (S \cup X)$. V prvem koraku algoritma je problem potujočega obiskovalca rešen kot primer problema trgovskega potnika na evklidski ravnini brez ovir. V naslednjem koraku približek iz prvega koraka razrešimo tako, da najdemo najkrajše poti med zaporednimi vozlišči obhoda iz prvega koraka. Pri tem uporabimo prilagojeni Floyd-Warshallov algoritem (angl. *Adapted Floyd-Warshall Algorithm*), prikazan s psevdokodo v algoritmu 11, ki najde najkrajše poti samo med vozlišči S in pri tem se lahko sprehodi tudi skozi X. V naslednjem koraku izdelamo iz matrike povezav W matriko povezav Z, dimenzij $(S \times S)$ ki predstavlja rešitev problema iskanja najkrajših poti vseh parov (angl. *All-Pairs Shortest Paths Problem (APSP)*, poglavje 2.1.5). Na koncu, v zanki (od vrstice 6 do 8) izračunamo skupno ceno obhoda za TVP.

Druga predlagana metoda za reševanje problema potujočega obiskovalca je algoritem Koper (angl. *Koper Algorithm*) prikazan s psevdokodo v algoritmu 12. Algoritem ima enake parametre kot naivni algoritem. V prvem koraku z uporabo prilagojenega Floyd-Warshallovega algoritma poiščemo najkrajše poti med vsemi pari vozlišč množice S v grafu G. Vhodno matriko razdalj označimo z W, izhodno matriko razdalj z Z. V naslednjem koraku, rešimo problem trgovskega potnika za matriko razdalj Z. Z rešitvijo problema TSP smo dobili obhod π, ki predstavlja rešitev za problem potujočega obiskovalca.

Za testiranje naše strategije smo uporabili primere problema potujočega obiskovalca, ki smo jih izdelali na osnovi uradnih turističnih zemljevidov: Kopra, Beograda in Benetk. Za Beograd smo izdelali primera, ki se razlikujeta po številu vozlišč v grafu. Poleg tega smo iz knjižnice TSPLIB vzeli primera *lin318* in *pr1002* ter ju ustrezno preoblikovali, da sta predstavljala problem potujočega obiskovalca. S tem smo dobili sintetičen a dovolj velik problem za testiranje. Izvedli smo poskuse za 5 primerov z različnimi velikostmi, ki znašajo od 120 do 1002 vozlišč za posamezen primer.

Za reševanje problema potujočega obiskovalca smo primerjali dve metodi: naivni algoritem, prikazan v algoritmu 10, in algoritem Koper, prikazan v algoritmu 12. Za reševanje TSP, ki je eden izmed korakov pri obeh algoritmih, smo uporabili algoritem Concorde, opisan v poglavju 2.2.5.

Rezultati poskusa so predstavljeni v tabeli 4.1. Peti stolpec tabele 4.1, predstavlja dolžino sprehoda in ustreza ceni rešitve, ki smo jo dobili pri poskusu. V vseh šestih primerih smo najkrajše sprehode pridobili z algoritmom Koper. V zadnjem stolpcu v tabeli 4.1 je prikazana relativna razlika med metodami v odstotkih. Algoritem Koper je v vseh testiranih primerih deloval boljše od naivnega algoritma. Kakovost rešitev je bila med 6,52 odstotka (v primeru Belgrade163) in 354,46 odstotka (v primeru pr1002) boljša z algoritmom Koper v primerjavi z naivnim algoritmom.

Namen tega raziskovanja je bil opisati in rešiti nov problem v teoriji grafov, imenovan problem potujočega obiskovalca. Čeprav je omenjeni problem podoben dobro znanemu problemu trgovskega potnika, so rezultati daleč od optimalnih, ko ga skušamo rešiti z naivnim algoritmom. V vseh testiranih primerih problema potujočega obiskovalca je algoritem Koper prekašal naivni algoritem.

5.5 Metodologija

Glavno orodje za opis in definicijo problema potujočega obiskovalca je teorija grafov. Za dejanske primere problema potujočega obiskovalca smo uporabili geografski informacijski sistem *Google Earth*, čigar podatkovno bazo smo uporabili za pridobitev mestnih znamenitosti, križišč in sprehajalnih poti. Pri izpeljavi in dokazovanju hipoteze 1 smo uporabili platformo za raziskovanje genetskih algoritmov *EA Visualizer* [15]. Aplikacija je napisana v programskem jeziku Java.

Pri reševanju hipoteze 2 smo uporabili Floyd-Warshallov algoritem [27] in ga ustrezno nadgradili za iskanje najkrajših poti samo med nekaterimi pari vozlišč grafa $G = (V, E, W)$. Uporabili smo tudi znane optimizacijske metode za reševanje simetričnega problema trgovskega potnika, presečne ravnine [108, 110], na osnovi katerega temelji metoda Concorde [3–5] in hevristične metode algoritma Lin-Kernighan [6, 45, 66, 75]. Programska orodja ki uporabljajo te metode so implementirane v programskem jeziku AnsiC.

Pri dokazovanju hipoteze 2 smo reševali tudi probleme linearnega programiranja (angl. *linear programming problems*) [22, 126] z uporabo simpleksne metode (angl. *simplex methods*) [31].

5.6 Doprinos k znanosti

Doprinos k znanosti predstavljajo naslednji rezultati:

- izdelava cepljenega hibridnega genetskega algoritma za reševanje problema trgovskega potnika,
- vpogled v različne stopnje ter mesta hibridizacije cepljenega genetskega algoritma,
- izdelava metode za reševanje problema potujočega obiskovalca,
- izdelava izboljšanega Floyd-Warshallovega algoritma za reševanje problema iskanja najkrajših poti vseh parov,
- rešitev problema potujočega obiskovalca za mesta: Koper, Beograd in Benetke.

Naj omenimo še, da so rezultati disertacije objavljeni v naslednjih znanstvenih člankih:

- M. Djordjevic, Influence of Grafting a Hybrid Searcher Into the Evolutionary Algorithm, *Proceedings of the 17th International Electrotechnical and Computer Science Conference, Portoroz, Slovenia* (2008), 115–118.
- M. Djordjevic, M. Tuba, and B. Djordjevic, Impact of Grafting a 2-opt Algorithm Based Local Searcher Into the Genetic Algorithm, *Proceedings of the 9th WSEAS international conference on Applied informatics and communications, AIC 2009, Moscow, Russia* (2009), 485–490.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Book of Abstract of International Conference on Operations Research, OR 2011, Zurich, Switzerland* (2011), 130.
- M. Djordjevic, and A. Brodnik, Quantitative Analysis of Separate and Combined Performance of Local Searcher and Genetic Algorithm, *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011, Dubrovnik, Croatia* (2011), 515–520.
- M. Djordjevic, M. Grgurovic and A. Brodnik, The Traveling Visitor Problem and the Koper Algorithm for Solving It, *Book of Abstracts of 25th Conference of European Chapter on Combinatorial Optimization, ECCO 2012, Antalya, Turkey* (2012), 10.
- M. Djordjevic, M. Grgurovic and A. Brodnik, The Traveling Visitor Problem and Algorithms for Solving It, *Book of Abstracts of 3rd Student Conference on Operational Research, SCOR 2012, Nottingham, UK* (2012), 26.
- M. Djordjevic, J. Zibert, M. Grgurovic, and A. Brodnik, Methods for Solving the Traveling Visitor Problem, *Proceedings of the 1st International Internet and Business Conference, IBC 2012, Rovinj, Croatia* (2012), 174–179.

- M. Djordjevic, M. Grgurovic, and A. Brodnik, Performance Analysis of Partial Use of Local Optimization Operator on Genetic Algorithm for Traveling Salesman Problem, *Business Systems Research*, Print ISSN 1847-8344; Online ISSN 1847-9375, in press.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications.* Prentice hall, Upper Saddle River (New Jersey), 1993.

[2] S. Anily and R. Hassin. The swapping problem. *Networks*, 22(4):419–433, 1992.

[3] D. Applegate, R. Bixby, V. Chvátal, and B. Cook. Finding cuts in the TSP. Technical report, Center for Discrete Mathematics and Theoretical Computer Science, Rutgers, 1995.

[4] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. TSP cuts which do not conform to the template paradigm. In M. Jünger and D. Naddef, editors, *Computational Combinatorial Optimization*, pages 261–304. Springer Verlag, Berlin, 2001.

[5] D. Applegate, R. Bixby, W. Cook, and V. Chvátal. *On the solution of traveling salesman problems.* Rheinische Friedrich-Wilhelms-Universitat Bonn, Bonn, 1998.

[6] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS Journal on Computing*, 15(1):82–92, 2003.

[7] T. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, New York, 1996.

[8] M. O. Ball and M. J. Magazine. Sequencing of insertions in printed circuit board assembly. *Operations Research*, 36(2):192–201, 1988.

[9] J. C. Bean. Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on computing*, 6(2):154–160, 1994.

[10] J. J. Bentley. Fast algorithms for geometric traveling salesman problems. *INFORMS Journal on Computing*, 4(4):387, 1992.

[11] A. Blum, P. Chalasani, D. Coppersmith, B. Pulleyblank, P. Raghavan, and M. Sudan. The minimum latency problem. In *Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 163–171, 1994.

[12] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[13] K. D. Boese. *Models for iterative global optimization.* PhD thesis, University of California, Los Angeles, 1996.

[14] N. Boland, L. Clarke, and G. Nemhauser. The asymmetric traveling salesman problem with replenishment arcs. *European Journal of Operational Research*, 123(2):408–427, 2000.

[15] P. Bosman and D. Thierens. On the modelling of evolutionary algorithms. In *Proceedings of the Eleventh Belgium–Netherlands Conference on Artificial Intelligence BNAIC*, pages 67–74, 1999.

[16] J. Brest and J. Žerovnik. A heuristic for the asymmetric traveling salesman problem. In *The 6th Metaheuristics International Conference*, pages 145–150, 2005.

[17] T. N. Bui and B. R. Moon. A new genetic approach for the traveling salesman problem. In *Evolutionary Computation, IEEE World Congress on Computational Intelligence, Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 7–12, 1994.

[18] G. Carpaneto, M. Dell'Amico, and P. Toth. Exact solution of large-scale, asymmetric traveling salesman problems. *ACM Transactions on Mathematical Software (TOMS)*, 21(4):394–409, 1995.

[19] A. E. Carter and C. T. Ragsdale. A new approach to solving the multiple traveling salesperson problem using genetic algorithms. *European journal of operational research*, 175(1):246–257, 2006.

[20] R. Cheng and M. Gen. Resource constrained project scheduling problem using genetic algorithms. *International Journal of Intelligent Automation and Soft Computing*, 3(3):78–286, 1997.

[21] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms - i. representation. *Computers & Industrial Engineering*, 30(4):983–997, 1996.

[22] V. Chvátal. *Linear programming.* WH Freeman, New York, 1983.

[23] J. Cirasella, D. Johnson, L. McGeoch, and W. Zhang. The asymmetric traveling salesman problem: Algorithms, instance generators, and tests. In *Revised Papers from the Third International Workshop on Algorithm Engineering and Experimentation*, ALENEX '01, pages 32–59, 2001.

[24] A. Coja-Oghlan, S. O. Krumke, and T. Nierhoff. A heuristic for the stacker crane problem on trees which is almost surely exact. *Journal of Algorithms*, 61(1):1–19, 2006.

[25] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158, 1971.

[26] W. Cook and P. Seymour. Tour merging via branch-decomposition. *INFORMS Journal on Computing*, 15(3):233–248, 2003.

[27] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms.* 3rd edition. The MIT Press, Cambridge, 2009.

[28] G. Cornuéjols, J. Fonlupt, and D. Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical Programming*, 33(1):1–27, 1985.

[29] G. A. Croes. A method for solving traveling-salesman problems. *Operations Research*, 6(6):791–812, 1958.

[30] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 2(4):393–410, 1954.

[31] G. B. Dantzig and W. Orchard-Hays. The product form for the inverse in the simplex method. *Mathematical Tables and Other Aids to Computation*, 8(46):64–67, 1954.

[32] S. Dasgupta, C. H. Papadimitriou, and U. Vazirani. *Algorithms.* 1st ed. McGraw-Hill, New York, 2008.

[33] F. Della Croce, R. Tadei, and G. Volta. A genetic algorithm for the job shop problem. *Computers & Operations Research*, 22(1):15–24, 1995.

[34] M. Djordjevic. Influence of grafting a hybrid searcher into the evolutionary algorithm. In *Proceedings of the 17th International Electrotechnical and Computer Science Conference*, pages 115–118. Slovenian Section IEEE, Ljubljana, 2008.

[35] M. Djordjevic and A. Brodnik. Quantitative analysis of separate and combined performance of local searcher and genetic algorithm. In *Book of Abstracts, OR 2011, International Conference on Operation Research*, page 130. IFOR, Zurich, 2011.

[36] M. Djordjevic and A. Brodnik. Quantitative analysis of separate and combined performance of local searcher and genetic algorithm. In *Proceedings of the 33rd International Conference on Information Technology Interfaces, ITI 2011*, pages 515–520. SRCE, Zagreb, 2011.

[37] M. Djordjevic, M. Tuba, and B. Djordjevic. Impact of grafting a 2-opt algorithm based local searcher into the genetic algorithm. In *Proceedings of the 9th WSEAS international conference on Applied informatics and communications*, pages 485–490. Stevens Point, WSEAS, 2009.

[38] M. Dorigo, M. Birattari, and T. Stutzle. Ant colony optimization. *IEEE Computational Intelligence Magazine*, 1(4):28–39, 2006.

[39] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[40] Z. Drezner. *Facility location: a survey of applications and methods.* Springer, Berlin, 1995.

[41] K. Easton, G. Nemhauser, and M. Trick. The traveling tournament problem description and benchmarks. In *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, CP '01, pages 580–584, 2001.

[42] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, and A. Battersby. Hybrid genetic algorithms: a review. *Engineering Letters*, 13(2):124–137, 2006.

[43] H. Emmons and K. Mathur. Lot sizing in a no-wait flow shop. *Operations research letters*, 17(4):159–164, 1995.

[44] C. Engels and B. Manthey. Average-case approximation ratio of the 2-opt algorithm for the tsp. *Operations Research Letters*, 37(2):83–84, 2009.

[45] T. Fischer and P. Merz. A distributed chained Lin-Kernighan algorithm for TSP problems. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, pages 162–172, 2005.

[46] B. Fleischmann. A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research*, 21(3):307–317, 1985.

[47] C. Fleurent and J. A. Ferland. Genetic hybrids for the quadratic assignment problem. *American Mathematical Society*, 16:173–187, 1993.

[48] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, 1956.

[49] L. J. Fogel. *Intelligence through simulated evolution: forty years of evolutionary programming.* John Wiley & Sons, New York, 1999.

[50] B. Freisleben and P. Merz. New genetic local search operators for the traveling salesman problem. In *Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, PPSN IV, pages 890–899, 1996.

[51] P. Gang, I. Iimura, and S. Nakayama. An evolutionary multiple heuristic with genetic local search for solving TSP. *International Journal of Information Technology*, 14(2):1–11, 2008.

[52] A. Garcia, P. Jodrá, and J. Tejel. A note on the traveling repairman problem. *Networks*, 40(1):27–31, 2002.

[53] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness.* WH Freeman, New York, 1979.

[54] F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1-3):223–253, 1996.

[55] M. X. Goemans and D. J. Bertsimas. Probabilistic analysis of the Held and Karp lower bound for the Euclidean traveling salesman problem. *Mathematics of operations research*, 16(1):72–89, 1991.

[56] D. E. Goldberg. *Genetic algorithms in search, optimization, and machine learning.* Addison-Wesley, Reading, 1989.

[57] L. Gouveia and S. Vob. A classification of formulations for the (time-dependent) traveling salesman problem. *European Journal of Operational Research*, 83(1):69–82, 1995.

[58] G. Gutin, H. Jakubowicz, S. Ronen, and A. Zverovitch. Seismic vessel problem. *Communications in Dependability and Quality Management*, 8(1):13–20, 2005.

[59] G. Gutin and D. Karapetyan. A memetic algorithm for the generalized traveling salesman problem. *Natural Computing*, 9(1):47–60, 2010.

[60] G. Gutin and A. P. Punnen. *The traveling salesman problem and its variations.* Kluwer Academic Publishers, Dordrecht, 2002.

[61] N. Guttmann-Beck, R. Hassin, S. Khuller, and B. Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000.

[62] M. Hahsler and K. Hornik. TSP–Infrastructure for the traveling salesperson problem. *Journal of Statistical Software*, 23(2):1–21, 2007.

[63] W. E. Hart. *Adaptive global optimization with local search.* PhD thesis, University of California, San Diego, 1994.

[64] L. He and N. Mort. Hybrid genetic algorithms for telecommunications network back-up routeing. *BT Technology Journal*, 18(4):42–50, 2000.

[65] M. Held and R.M. Karp. The traveling-salesman problem and minimum spanning trees: Part II. *Mathematical Programming*, 1(1):6–25, 1971.

[66] K. Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[67] J. H. Holland. *Adaptation in natural and artificial systems.* The University of Michigan Press, Ann Arbor, 1975.

[68] H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications.* Morgan Kaufmann Publishers, San Francisco, 2004.

[69] L. J. Hubert and F. B. Baker. Applications of combinatorial programming to data analysis: the traveling salesman and related problems. *Psychometrika*, 43(1):81–91, 1978.

[70] D. Johnson and L. McGeoch. Experimental analysis of heuristics for the STSP. In *The traveling salesman problem and its variations*, pages 369–443. Kluwer Academic Publishers, Dordrecht, 2004.

[71] D. S. Johnson and L. A. McGeoch. The traveling salesman problem: a case study in local optimization. In *Local search in combinatorial optimization*, pages 215–310. J. Wiley, Chichester, 1997.

[72] D. S. Johnson and C. H. Papadimitriou. *Computational complexity and the traveling salesman problem.* Massachusetts Institute of Technology, Cambridge, 1981.

[73] S. Jung and B. R. Moon. The natural crossover for the 2d euclidean tsp. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pages 1003–1010. Morgan Kaufmann, San Mateo, 2000.

[74] M. Junger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. *Handbooks in Operations Research and Management Science*, 7:225–330, 1995.

[75] D. Karapetyan and G. Gutin. Lin-Kernighan heuristic adaptations for the generalized traveling salesman problem. *European journal of operational research*, 208(3):221–232, 2011.

[76] R. M. Karp and C. H. Papadimitriou. On linear characterizations of combinatorial optimization problems. *SIAM Journal on Computing*, 11:610–620, 1982.

[77] K. Katayama and H. Narihisa. Iterated local search approach using genetic transformation to the traveling salesman problem. In *Proceedings of GECCO'99*, pages 321–328, 1999.

[78] K. Katayama, H. Sakamoto, and H. Narihisa. The efficiency of hybrid mutation genetic algorithm for the travelling salesman problem. *Mathematical and Computer Modelling*, 31(10-12):197–203, 2000.

[79] R. Keuthen. *Heuristic Approaches for Routing Optimisation.* PhD thesis, University of Nottingham, Nottingham, 2003.

[80] S. Kirkpatrick. Optimization by simulated annealing: quantitative studies. *Journal of Statistical Physics*, 34(5):975–986, 1984.

[81] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671, 1983.

[82] J. R. Koza. *Genetic programming: On the programming of computers by natural selection.* MIT Press, Cambridge, 1992.

[83] N. Krasnogor. *Studies on the theory and design space of memetic algorithms.* PhD thesis, University of the West of England, Bristol, 2002.

[84] N. Krasnogor, P. Moscato, and M. G. Norman. A new hybrid heuristic for large geometric traveling salesman problems based on the delaunay triangulation. In *Anais do XXVII Simposio Brasileiro de Pesquisa Operacional, Vitoria, Brazil, SOBRAPO*, pages 6–8, 1995.

[85] N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *Evolutionary Computation*, 9(5):474–488, 2005.

[86] K. W. C. Ku and M. W. Mak. Empirical analysis of the factors that affect the baldwin effect. *Lecture notes in computer science*, pages 481–490, 1998.

[87] A. H. Land and A. G. Doig. An automatic method of solving discrete programming problems. *Econometrica*, 28(3):497–520, 1960.

[88] J. K. Lenstra and A. H. G. R. Kan. Some simple applications of the travelling salesman problem. *Operational Research Quarterly*, 26(4):717–733, 1975.

[89] C. F. Liaw. A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124(1):28–42, 2000.

[90] G. F. Lima, A. S. Martinez, and O. Kinouchi. Deterministic walks in random media. *Physical Review Letters*, 87(1):10603, 2001.

[91] S. Lin and B. W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations research*, 21(2):498–516, 1973.

[92] S. B. Liu, K. M. Ng, and H. L. Ong. A new heuristic algorithm for the classical symmetric traveling salesman problem. *International Journal of Mathematical Science*, 37(4):234–238, 2007.

[93] V. Mak and N. Boland. Heuristic approaches to the asymmetric travelling salesman problem with replenishment arcs. *International Transactions in Operational Research*, 7(4-5):431–447, 2000.

[94] K. Menger. Das botenproblem. *Ergebnisse eines Mathematischen Kolloquiums*, 2:11–12, 1932.

[95] M. Mernik, V. Žumer, and M. Črepinšek. A metaevolutionary approach for the traveling salesman problem. In *Information Technology Interfaces, ITI 2000, Proceedings of the 22nd International Conference*, pages 357–362, 2000.

[96] P. Merz. *Memetic algorithms for combinatorial optimization problems: Fitness landscapes and effective search strategies.* PhD thesis, University of Siegen, Siegen, 2000.

[97] P. Merz and B. Freisleben. Genetic local search for the TSP: new results. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 159–164, 1997.

[98] P. Merz and B. Freisleben. Memetic algorithms for the traveling salesman problem. *Complex Systems*, 13(4):297–346, 2001.

[99] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a lamarckian genetic algorithm and an empirical binding free energy function. *Journal of computational chemistry*, 19(14):1639–1662, 1998.

[100] R. E. Mowe and B. A. Julstrom. A web-based evolutionary algorithm demonstration using the traveling salesman problem. In *The 34th Annual Midwest Instruction and Computing Symposium*, pages 1–10, 2001.

[101] Y. Nagata. Edge assembly crossoveria high-power genetic algorithm for the traveling salesman problem. In *Proceedings of the 7th International Conference on Genetic Algorithms*, page 450. Morgan Kaufmann Publishers, San Francisco, 1997.

[102] H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga. Greedy genetic algorithms for symmetric and asymmetric tsps. *Joho Shori Gakkai Shinpojiumu Ronbunshu*, 2001(12):67–74, 2001.

[103] H. D. Nguyen, I. Yoshihara, K. Yamamori, and M. Yasunaga. Implementation of an effective hybrid GA for large-scale traveling salesman problems. *Systems, Manand Cybernetics*, 37(1):92–99, 2007.

[104] C. E. Noon and J. C. Bean. An efficient transformation of the generalized traveling salesman problem. In *Technical report.* University of Michigan, Ann Arbor, 1989.

[105] CS Orloff. A fundamental problem in vehicle routing. *Networks*, 4(1):35–64, 1974.

[106] A. J. Orman and H. P. Williams. A survey of different integer programming formulations of the travelling salesman problem. In *Optimisation, Econometric and Financial Analysis*, pages 91–104. Springer, Berlin, 2007.

[107] E. Ozcan and M. Erenturk. A brief review of memetic algorithms for solving Euclidean 2D traveling salesrep problem. In *Proceedings of the 13th Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 99–108, 2004.

[108] M. Padberg and M. Grötschel. Polyhedral computations. In *The Traveling Salesman Problem*, pages 307–360. John Wiley & Sons, Chichester, 1985.

[109] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

[110] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[111] S. S. Ray, S. Bandyopadhyay, and S. K. Pal. Genetic operators for combinatorial optimization in TSP and microarray gene ordering. *Applied Intelligence*, 26(3):183–195, 2007.

[112] I. Rechenberg. Evolution strategy. In *Computational Intelligence: imitating life*, pages 147–159. IEEE, New York, 1994.

[113] C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers & Operations Research*, 22(1):5–13, 1995.

[114] C. Rego. Relaxed tours and path ejections for the traveling salesman problem. *European Journal of Operational Research*, 106(2-3):522–538, 1998.

[115] C. Rego and F. Glover. Local search and metaheuristics. In *The traveling salesman problem and its variations*, pages 309–368. Kluwer Academic Publishers, Dordrecht, 2004.

[116] G. Reinelt. Tsplib-a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[117] G. Reinelt. *The traveling salesman: computational solutions for TSP applications.* Springer-Verlag, Berlin, 1994.

[118] A. Schrijver. On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68, 2005.

[119] V. Sels and M. Vanhoucke. A hybrid dual-population genetic algorithm for the single machine maximum lateness problem. In *Evolutionary Computation in Combinatorial Optimization*, volume 6622, pages 14–25. Springer Verlag, New York, 2011.

[120] H. Sengoku and I. Yoshihara. A fast TSP solver using GA on Java. In *3rd International Symposium on Artificial Life and Robotics (AROB III'98)*, 1998.

[121] W. Spears, K. De Jong, T. Bäck, and D. Fogel. An overview of evolutionary computation. In *Machine Learning: ECML-93*, pages 442–459. Springer, Heidelberg, 1993.

[122] K. Steiglitz and P. Weiner. Some improved algorithms for computer solution of the traveling salesman problem. In *Proceedings of 6th Annual Allerton Conference on Circuit and System Theory*, pages 814–821. University of Illinois, Urbana, 1968.

[123] T. Stützle and M. Dorigo. Aco algorithms for the traveling salesman problem. *Evolutionary Algorithms in Engineering and Computer Science*, pages 163–183, 1999.

[124] A. Uğur, S. Korukoğlu, A. Çalışkan, M. Cinsdikici, and A. Alp. Genetic algorithm based solution for TSP on a sphere. *Mathematical and Computational Applications*, 14(3):219–228, 2009.

[125] N. Ulder, E. Aarts, H. J. Bandelt, P. van Laarhoven, and E. Pesch. Genetic local search algorithms for the traveling salesman problem. In *Parallel problem solving from nature*, pages 109–116. Springer Verlag, Berlin, 1991.

[126] R. Vanderbei. Linear programming: foundations and extensions. *Journal of the Operational Research Society*, 49(1):93–98, 1998.

[127] M. Vazquez and L. D. Whitley. A hybrid genetic algorithm for the quadratic assignment problem. In *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pages 135–142. Morgan Kaufmann, San Mateo, 2000.

[128] M. B. Wall. *A Genetic Algorithm for Resource-Constrained Scheduling.* PhD thesis, Massachusetts Institute of Technology, Massachusetts, 1996.

[129] S. Wang, B. Wang, and X. Li. Grafted genetic algorithm and its application. In *7th International Conference on Computer-Aided Industrial Design and Conceptual Design, CAIDCD'06*, pages 1–4, 2007.

[130] C. M. White and G. G. Yen. A hybrid evolutionary algorithm for traveling salesman problem. *Evolutionary Computation*, 2(1):1473–1478, 2004.

[131] M. Yamamura, T. Ono, and S. Kobayashi. Character-preserving genetic algorithms for traveling salesman problem. *Journal-Japanese Society For Artificial Intelligence*, 7:1049–1049, 1992.

[132] M. Zachariasen and M. Dam. Tabu search on the geometric traveling salesman problem. In *Meta-Heuristics: Theory and Applications*, volume 36, pages 571–587. Kluwer Academic Publishers, Boston, 1996.

[133] F. Zhao, J. Sun, S. Li, and W. Liu. A hybrid genetic algorithm for the traveling salesman problem with pickup and delivery. *International Journal of Automation and Computing*, 6(1):97–102, 2009.

[134] G. Zhao, W. Luo, H. Nie, and C. Li. A genetic algorithm balancing exploration and exploitation for the travelling salesman problem. In *4th International Conference on Natural Computation*, pages 505–509, 2008.

Index

Declaration

I declare that this PhD Thesis does not contain any materials previously published or written by another person except where due reference is made in the text.

Milan Djordjevic